NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CAPABILITY DELIVERY WITH FOG OF EMERGENCE

by

Wen Chong Julian Chow

September 2013

Thesis Co-Advisors:	Gary O. Langford Man-Tak Shing
Second Reader:	Robert C. Harney

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2013	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
CAPABILITY DELIVERY WITH FOG OF EMERGENCE

5. FUNDING NUMBERS

6. AUTHOR(S) Wen Chong Julian Chow

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943–5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release;distribution is unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

A proposed capability delivery ontology with fog of emergence provides a language construct to relate how the processes and parts of a notional capability delivery system incrementally produce and refine a capability through well-known life cycle phases. The natural propensity for capability delivery organizations to perform these life cycle activities using intended missions and requirements instead of as-deployed missions and emergent traits give rise to the fog of emergence that obscures the organizations perception of the capability as it is taken through its life cycle. Through capability delivery ontology, the embedded fog of emergence is used as a prism to separate the white light of capability performance into its constituent colors of "as needed," "as-planned," "as-known," and "as-deployed" perceived by the capability delivery organizations.

14. SUBJECT TERMS Capability delivery system, Emergence, Capability Delivery Ontology, Fog of Emergence

15. NUMBER OF PAGES
171

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01–280–5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

CAPABILITY DELIVERY WITH FOG OF EMERGENCE

Wen Chong Julian Chow
Civilian, Defence Science & Technology Agency, Singapore
B.Comp (Information Systems), National University of Singapore, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
September 2013

Author: Wen Chong Julian Chow

Approved by: Gary O. Langford, PhD
Thesis Co-Advisor

Man-Tak Shing, PhD
Thesis Co-Advisor

Robert C. Harney, PhD
Second Reader

Clifford A. Whitcomb
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A proposed capability delivery ontology with fog of emergence provides a language construct to relate how the processes and parts of a notional capability delivery system incrementally produce and refine a capability through well-known life cycle phases. The natural propensity for capability delivery organizations to perform these life cycle activities using intended missions and requirements instead of as-deployed missions and emergent traits give rise to the fog of emergence that obscures the organizations perception of the capability as it is taken through its life cycle. Through capability delivery ontology, the embedded fog of emergence is used as a prism to separate the white light of capability performance into its constituent colors of “as needed,” “as-planned,” “as-known,” and “as-deployed” perceived by the capability delivery organizations.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AoA	Analysis of Alternatives
APB	Acquisition Program Baseline
COCOM	Combatant Command
CDD	Capability Development Document
CDR	Critical Design Review
CONOPS	Concepts of Operations
CPD	Capability Production Plan
CTE	Critical Technology Element
DAMS	Defense Acquisition Management System
DoD	Department of Defense
DoDAF	Department of Defense Architectural Framework
DOTmLPF-P	Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel, Facilities and Policy
DT&E	Developmental Test & Evaluation
EMD	Engineering & Manufacturing Development
EMMI	Energy, Matter, Material Wealth, and Information
FCB	Functional Capabilities Board
FOC	Full Operational Capability
FOT&E	Follow-On Test & Evaluation
FRP	Full Rate Production
ICD	Initial Capability Document
INCOSE	International Council on Systems Engineering
IOC	Initial Operational Capability
IOT&E	Initial Operational Test & Evaluation
JCB	Joint Capabilities Board
JCIDS	Joint Capabilities Integration and Development System
JEON	Joint Emergent Operational Need
JROC	Joint Requirements Oversight Council
JUON	Joint Urgent Operational Need
LCSP	Life cycle Sustainment Plan

LRIP	Low Rate Initial Production
MDA	Milestone Decision Authority
MDD	Materiel Development Decision
MoE	Measures of Effectiveness
MSA	Materiel Solution Analysis
O&S	Operations & Support
P&D	Production & Development
PEO	Program Executive Officer
PDR	Preliminary Design Review
QDR	Quadrennial Defense Review
RAM	Reliability, Availability, and Maintainability
RFP	Request for Proposal
SE	Systems Engineering
SEP	Systems Engineering Plan
SEBoK	Systems Engineering Body of Knowledge
SoI	System of Interest
SoS	System of Systems
TD	Technology Development
TDS	Technology Development Strategy
TRL	Technology Readiness Level

EXECUTIVE SUMMARY

Since the 2000 Quadrennial Defence Review, the Department of Defense (DoD) has been re-orienting force development processes to the identification and support of user capabilities, with an emphasis on agile compositions of systems to meet a range of changing user needs.

Emergence is when a system does something that no subset of its parts can do, and these emergent traits exhibited by systems and system of systems (SoS) are tapped as military capabilities. The various architects, builders, and users (collectively referred to as capability delivery organizations) associated with the capability delivery's life cycle should be aware that emergent traits could be intentional, unintentional, desirable, or undesirable. Emergent traits beyond those desired as requirements continue to manifest when a system is deployed even if capability delivery organizations do not perceive it.

This thesis analyzed a notional capability delivery system (CDS) that takes a capability conceived as a need through its life cycle till its retirement. Both black box and white box approaches were adopted to analyze the input, output and noise factors the CDS were subjected to by the Joint Capabilities Integration and Development System (JCIDS) and to understand the parts and processes that makes the CDS work respectively. A new capability delivery ontology with a central theme of emergence was proposed after combining insights from literature on the philosophical, axiomatic and methodological perspectives of emergence and a Vitech CORE working implementation of the DoD Architecture Framework.

The CDS ontology and fog of emergence provide a language construct to relate how the processes facilitate the interaction of the parts of a CDS to incrementally produce and refine a capability through well-known DoD 5000.02 life cycle phases. The life cycle phases were mapped to a generic problem solving process of "analyze-design-build-test," where analysis produces/refines

the operational architecture, design produces/refines the system architecture, build verifies system components to the system architecture, and test validates system components to the operational architecture. The natural propensity for capability delivery organizations to perform these activities using intended missions and requirements instead of as-deployed missions and emergent traits give rise to the fog of emergence that obscures the organizations perception of the capability as it is taken through its life cycle.

Through capability delivery ontology, the embedded fog of emergence could be used as a prism to separate the white light of capability performance into its constituent colors of “as needed,” “as-planned,” “as-known” and “as-deployed” perceived by the capability delivery organizations.

The tractability of the ontology was demonstrated through a partial implementation of a capability delivery system simulator that embodied the concepts put forward by the ontology to step through capability delivery from cradle to grave according to DoD 5000.02 life cycle phases while subjected to input and noise factors from JCIDS.

This research sets a potential stage for further exploration into developing experiments toward understanding effects of input and control factors to capability delivery and eventually developing a normative model of capability delivery with emergence.

ACKNOWLEDGMENTS

This thesis is not possible without the patience, guidance, and support from my thesis Advisor, Professor Gary O. Langford. My perspective of Systems Engineering as a discipline has been greatly expanded in terms of horizon and resolution through our invaluable discussions on the thesis and Systems Engineering on the whole. It was my privilege to have been advised by Professor Gary O. Langford, who is very much the front of Systems Engineering knowledge with his varied and successful career as a lecturer, a practitioner, and now, a strategic re-visioner.

I would also like to thank my thesis co-advisor, Professor Man-Tak Shing for his guidance and encouragement. His sharp eye for details helped establish the link between the axiomatic knowledge expounded in the thesis and the exploratory software model developed. I would like to extend my gratitude to my second reader, Professor Robert C. Harney, for the feedback given to me.

Many friends from the Naval Postgraduate School have also contributed in one way or another towards the completion of this thesis, and I would like to especially thank Leo, Winson, and Zhi Feng for their friendship and support that made the enjoyable course in the school even more rewarding.

Finally, I would like to express my thanks to my family, especially to my wife, Ee Wean. It is certainly not easy being the wife to someone who has track-record of sacrificing leisure for work. Thank you for being so accommodating and this thesis is the fruit of the love, support and encouragement you have given me that kept me going.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. CAPABILITY DELIVERY NEEDS

The Department of Defense (DoD) has been re-orienting force development processes to the identification and support of user capabilities, with an emphasis on agile composition of systems to meet a range of changing user needs since the 2000 Quadrennial Defense Review (QDR) (Dahmann, Rebovich, & Lane, 2008).

A capability is the ability to achieve a desired effect under specified standards and conditions through combinations of ways and means to perform a set of activities (Deputy Chief Information Officer, 2010). A capability forms the basis of operational activities desired by users, which when carried out allows the users to achieve their missions.

B. SYSTEM OF SYSTEMS AND ASSOCIATED CHALLENGES

According to the Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering (ODUSD [A&T] SSE) (2008), there was an increasing number of military capabilities being achieved through a system of systems (SoS) approach. An SoS is "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities." The SoS-level capabilities are implemented by intended and desired emergent SoS-level traits associated with the SoS-level functions that arise due to the interactions and integration of constituent systems (Langford, 2013a).

As Rechtin (1991) described elegantly, emergence is when the system does something that no subset of its parts can do. Based on this definition, it follows that emergent traits could be intentional or unintentional, desirable or undesirable. In the modern day context, intended and desirable emergent traits of systems and SoS are tapped as military capabilities; however, the various architects, builders, and users (hereafter referred to as capability delivery

organizations) associated with the capability delivery's life cycle should be aware that unintended or undesired emergent properties and traits would also be present.

A more technical definition consistent with Rechtin's definition is that "emergence is any effect that produces a change in intrinsic properties, traits or attributes resulted by combining objects through interactions of objects with energy, matter, material wealth and information" (EMMI) (Langford, 2012). Simply put, emergence is a condition that exists when there is a change in exhibited traits of a constituent system in the context of an interaction between a pair of constituent systems.

The complexity of an SoS scales up faster than the increase in the number of constituent systems due to the number of possible interactions between constituent systems (Huynh & Langford, 2009), exacerbated by the fact that different emergent traits could result from the same interaction realized through different interfaces. Emergence serves to add uncertainty to the eventual achievable performance of the SoS during capability delivery.

While the increased number of constituent systems increases the uncertainty of system performance, the sheer length of the life cycle for an SoS-class of systems also increases the susceptibility of the SoS to changes in user needs precipitated by the changing face of war.

The QDR (U.S. Department of Defense [USDoD], 2001) recognized these challenges with its stated purpose to re-orient force development with an emphasis on composing an SoS in an agile manner and to meet a range of changing user needs. This recognition of agility in acquisition translates to a need for the SoS to be able to either deliver new capabilities using constituent systems or to expand its SoS boundaries to incorporate capabilities from other systems (legacy and new) to satisfy changing user needs.

C. RESEARCH QUESTIONS

This thesis focuses on developing an ontology to model the meta-system for capability development and delivery, which has been put in place in response to the capability-based approach to modernize the military, and that could be used as a handle to explore the effectiveness of the meta-system.

Research questions are developed around these concepts and modeling of the effectiveness of capability delivery.

- What is a capability delivery system (CDS)?
- How can this meta-system of capability delivery be modeled?
    - What are the input and noise factors to the CDS?
    - What are the parts and processes that comprise the CDS?
    - What is emergence and how can its effects be modeled?
    - What are the measures of effectiveness for the CDS?
    - What are the effects of the choice of SE process models, stability of capability needs, and capability complexity on the effectiveness of the CDS?
- What is an existing ontology suitable to describe capability delivery?
- In what ways can the existing CDS ontology be improved to include the fog of emergence?

D. RESEARCH CONTRIBUTION

The thesis is intended to provide the following contributions:

- **Capability delivery ontology with emergence** – The extension of capability delivery ontology with a central theme of emergence.
- **Capability delivery system simulator** – Development and implementation of a functional simulator of a capability delivery system based on the new ontology to demonstrate the tractability of the ontology and as well as threats to the validity of the ontology.
- **Measures of effectiveness for capability delivery** – Using the capability delivery ontology as a prism to separate the white light of capability performance into its constituent colors of “as needed,” “as-planned,” “as-known” and “as-deployed.” The capability delivery effectiveness is measured as the ability of the capability

delivery system to minimize the gaps between “as-needed,’ “as-planned,” “as-known” and “as-deployed.”

E. THESIS ROADMAP

The roadmap of this thesis is as follows:

Chapter II presents the literature review of five key concepts: (1) capability-based approach; (2) systems and SoSs; (3) emergence in systems; (4) system life cycles with SE process models; and (5) an ontology for capability delivery. First, while the intricacies of the capability-based approach are not the focus of this thesis, understanding the mechanisms for the approach helps show the logical consequence of the increased likelihood that systems or SoSs would have to respond to changes or insertions of capability needs. Second, the management challenges in responding to unstable needs would be different depending on whether the capability implementing System-of-Interest (SoI) is a system or an SoS. Third, the complexity in managing the SoI implementation is exacerbated due to a fog of emergence that creates a gap between the subjective perception of emergent traits by capability delivery organizations and the emergent traits’ objective manifestation. Many SE process models exist to provide a guiding hand for capability delivery organizations to take a capability through its life cycle translating capability needs into operational capabilities. The system life cycle and SE process models do form the fourth part of the review. The final and fifth piece of the literature review is to present an existing capability delivery ontology that is familiar to readers who know the DoD Architectural Framework (DoDAF).

Chapter III covers the research approach to answer the research objectives. It covers how the capability delivery system with emergence is developed, and how it would be modeled to explore the research objectives.

Chapter IV describes the concept behind the capability delivery simulator that was developed for the purpose of exploring the new capability delivery model with emergence.

Chapter V describes the preliminary software design of the capability delivery simulator.

Chapter VI provides the summary of the implementation of an exploratory capability delivery system simulator, its features, shortcomings and how it could be used for future experiments based on the proposed capability delivery ontology with fog of emergence.

Chapter VII highlights the research contributions and concludes with the use of the proposed capability delivery ontology with fog of emergence to reflect on the thesis journey to deliver a capability delivery system simulator.

THIS PAGE INTENTIONALLY LEFT BLANK

II. LITERATURE REVIEW

The *Systems Engineering Guide for Systems of Systems (SE Guide for SoS)* (ODUSD [A&T] SSE, 2008) states that there are an increasing number of military capabilities being implemented through an SoS approach. An SoS is "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities," according to the *SE Guide for SoS*.

Similarly, the *Systems Engineering Body of Knowledge (SEBoK)* (Pyster & Olwell, 2013) acknowledges that most practitioners recognize a strong relationship between capability and SoS; however, there is no agreed position with regard to that relationship. There are two widely accepted views: the first describes the relationship as that of composition whereby a capability comprises a range of systems, processes, people, information and organization; the second describes the relationship as that of a property whereby capability is an emergent property of SoS. This author prefers the second relationship and in the following sections shows that the second relationship is more broadly applicable and, in fact, encompasses the spirit of the first.

In order to develop or extend an ontological model of the capability delivery meta-system, we have to unravel the relationship between capability delivery and SoS, and then understand what makes it work.

The following sections examine the literature concerning key concepts and expand definitions that are consistent and fit for the purpose of developing the model. We shall specifically look at the following concepts:

- Capability-Based Approach
- Systems and System of Systems
- Emergence in Systems
- Systems Engineering Life cycle and Process Models
- Capability Delivery Ontology

A. CAPABILITY-BASED APPROACH

A capability is the ability to achieve a desired effect under specified standards and conditions through combinations of ways and means to perform a set of tasks (DCIO, 2010).

Schrader, Lewis, & Brown (2003) review the lessons on managing change in the U.S. DoD based on two earlier QDRs performed in 1997 and 2001. The primary motivation for such reviews has been to ensure there were sufficient forces to execute strategies relevant to the projected threats. Schrader et al. (2003) state that prior to the QDRs, a "mismatch" between defense strategy and resource allocation was already recognized. A key recommendation from QDR 2001 was to adopt a capabilities-based strategy with senior military leadership assisting the U.S. Secretary of Defense in making balanced trade-offs that cut across services. This recommendation would allow Congress to prioritize future capabilities and provide guidance on forces, resources and pace of change.

The *JCIDS Manual*, 2012 describes "detailed guidelines and procedures for operation of the Joint Capabilities Integration and Development System (JCIDS) and interactions with several other departmental processes to facilitate the timely and cost effective development of capability solutions to the warfighter" (p. 1). The JCIDS deliberate staffing process and urgent staffing processes are shown in Figure 1 and Figure 2, respectively.

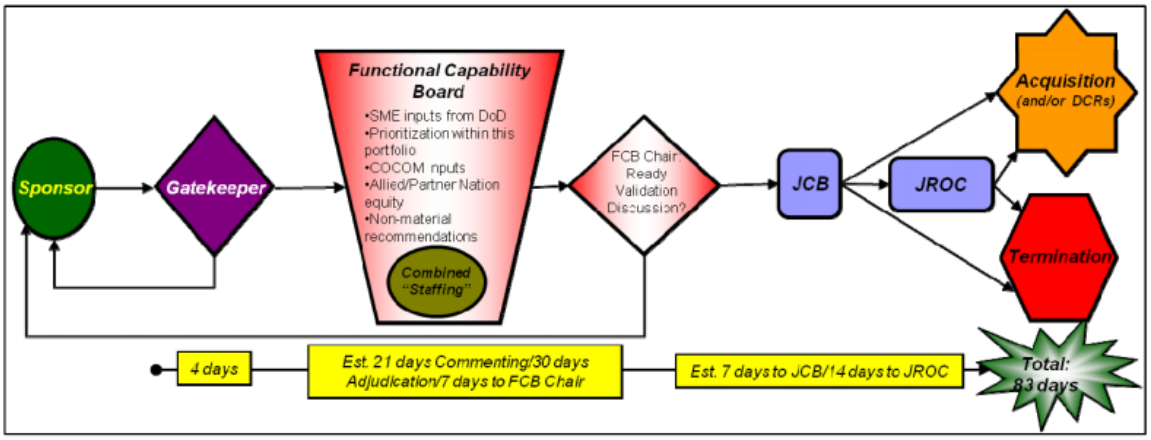


Figure 1. JCIDS deliberate staffing process (From *JCIDS Manual*, 2012).

The JCIDS deliberate staffing process should take no longer than 83 calendar days from the time a Sponsor submits a document identifying a capability gap to the Gatekeeper for review.

- The Gatekeeper supports the activities of the JCIDS process and manages the flow of documents in and out of the process (*JCIDS Manual*, 2012). The Gatekeeper would assign the document to the relevant lead and supporting Functional Capabilities Board (FCBs) within four days.
- The FCBs assesses the document to compare capability requirements to existing capability requirements, development programs, and fielded solutions within their respective portfolios (*JCIDS Manual*, 2012). The review considers partial or whole non-materiel changes to requirements and partner collaboration advice. The assessment would be made available to Services, Combatant Commands (COCOMs), and other DoD components for their comments by the end of 21 days (*JCIDS Manual*, 2012).
- The Sponsor has 30 days to satisfactorily adjudicate comments received, after which the FCB has 7 days to review the changes and to assist the FCB chair in making a validation decision (*JCIDS Manual*, 2012). A valid recommendation bears the certification by the FCB chair that the proposed capability solution is not redundant to existing capabilities (*JCIDS Manual*, 2012).
- The validation authorities could be either the Joint Capabilities Board (JCB).or Joint Requirements Oversight Council (JROC) depending on relevant level of interest. The JCB is a board below the JROC. The validation authorities should not take more than 21 calendar days to reach a decision after the FCB chair submits a

valid recommendation. The decision would be to either terminate the recommended capability, to begin acquisition or to execute DOTmLPF-P[1] Change Recommendations.

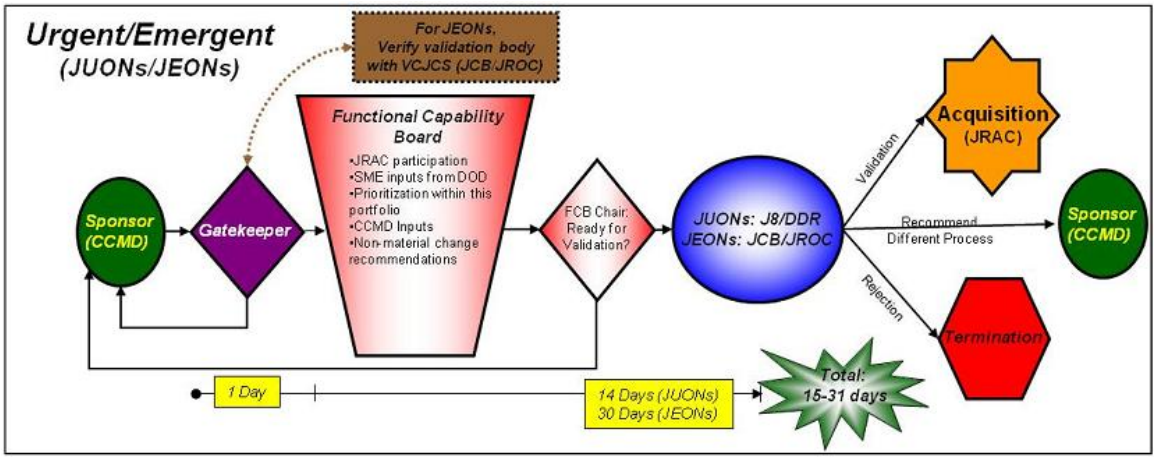


Figure 2. JCIDS urgent/emergent staffing process (From *JCIDS Manual*, 2012).

When a capability COCOM requirement is deemed as a joint urgent or emergent operational need (JUON or JEON, respectively), the staffing process as shown in Figure 2 could be used to expedite validation (*JCIDS Manual*, 2012). The validation process is expected to take no longer than 15 days and 31 days respectively for JUON and JEON (*JCIDS Manual*, 2012).

A summary of the JCIDS as gleaned from the *JCIDS Manual* (2012) can be summarized as:

> In its simplest, the capabilities-based strategy provides a strategic oversight that matches capability providers with users. The strategic oversight recognizes the importance to strike a balance between the instability of user requirements precipitated by the changing face of war and the need to provide stable intermediate forms of military capabilities to facilitate implementation accountability and better return on investment. New user needs would have to be validated in terms of whether any unacceptable loss of life or critical mission failure would be incurred should the need be left unaddressed. A validated need would then be assessed against the capability portfolio to determine if the need

[1] DOTmLPF-P stands for Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel, Facilities and Policy.

> could be satisfied by any existing capability provider or has to be satisfied through the establishment of a new capability. It is also possible for the validated need to be satisfied through a mixture of new and existing capabilities.

For example, a sponsor with a capability gap that could not be satisfied by capabilities in the Joint Forces could initiate a change recommendation to establish the new capability solution within the sponsor organization. Subsequently, a second sponsor with the same capability gap could put in a request for forces to leverage on the existing capability solution without reinventing the wheel. A third sponsor with the same capability gap that has to be organically incorporated could then generate a joint change request to bring the capability solution into its own organization.

The main benefits of the capability-based delivery are the greater strategic involvement of senior military leadership and Congress in directing and managing how temporally unstable user needs are satisfied by agilely composed capability solutions. It also follows that a better return on investment would be achieved through the use of existing capability solutions either in part or whole to service new capability needs.

The main implication of the JCIDS on the CDS is that it could be modeled as a source of either input or noise factors. If a valid capability need establishes a new capability solution, this is a new capability need into the CDS as an input factor. If a valid capability need is matched with an existing capability solution, the capability need is inserted to the CDS as noise factors. With the focus on this capability-based approach, it is more likely than before that a CDS would be subjected to unstable capability needs while a capability is in the process of being delivered.

B. SYSTEMS AND SYSTEM OF SYSTEMS

1. System

Maier & Rechtin (2009) defined a system as “a collection of things or elements that, working together, produce a result not achievable by the things

alone." The *DoD Architectural Framework (DoDAF) Glossary* (DCIO, 2010) defined a system as "a functionally, physically, and/or behaviorally related group of regularly interacting or interdependent elements."

The first two definitions emphasize the notion that a system is composed of elements. These elements could be functionally, physically or behaviorally related. These elements interact regularly to produce a result not achievable by the elements alone.

Langford (2013a) defines a system as:

> A group of adaptively stable and agile objects showing intrinsic emergence based on interactions with other objects. The condition for systemic behavior is a non-reciprocal change in boundary conditions of the objects resulting in a change in the properties of the objects. Systems are comprised of objects and processes.

This third definition by Langford (2013a) is used in this thesis because it is both abstract enough to encompass classical definitions of a system, while still precise enough for a practitioner to use as a litmus test differentiating a system from its parts. The third definition is consistent with the earlier two definitions and then goes on further to introduce the following qualifying conditions that must be satisfied for a system (Langford 2013a):

- Composition. A system is comprised of objects and processes.
- Agile adaptation. The objects adapt their properties, traits or attributes with each other through agile interactions.
- Stable adaptation. This interaction causes some degree of permanence and stability in the adapted properties, traits or attributes of a proper subset of objects of the system. Stability is maintained through dynamic adjustments about a point that falls within a region of stability. In other words, there are regions of exchanges between system elements where EMMI use is self-sustaining.
- Non-reciprocal emergence. If the observed adapted properties, traits or attributes of these stable and agile objects (manifested as changes in the conditions of the objects' functional, physical and behavioral boundaries) are non-reciprocal between their existence as a whole and existence as individual parts, we have emergence.

Hitchins (2000) said that “systems engineering appears to be all things to all people” and proposed a five-layer model for systems engineering that attempted to bring the divergence of SE as a practice under a common model:

- **Layer 5** – Socio-economic. The stuff of regulation and government control.
- **Layer 4** – Industrial Systems Engineering or engineering of complete supply chains/circles. Many industries make a socio-economic system.
- **Layer 3** – Business Systems Engineering. Many businesses make an industry. At this level, systems engineering seeks to optimize performance somewhat independent of other businesses.
- **Layer 2** – Project or System Level. Many projects make a Business. Western engineer-managers operate at this level, principally making complex artifacts.
- **Layer 1** – Product Level. Many products make a system. The tangible artifact level. Many engineers and their institutions consider this to be the only “real” systems engineering.

Hitchins (2000) points out that the statements associated with the five layers are approximate, but they serve to illustrate that systems could be nested with each lower layer contributing to the one above. Hitchins’ model showed that the methods to be employed by the systems engineer vary depending on the layer of interest, or here stated as the level of abstraction.

Keet, 2008 extended the concept of nesting further by introducing the concept of granularity in which granules (objects and processes) could also be partitioned heterarchically. A heterarchy is a system of organization replete with overlap, multiplicity where each element shares the same horizontal positional relationship. An important characteristic of heterarchical granularity is that these granules may overlap in a self-adjudicated manner appropriate to the context in which the relationship exists (Langford, 2012). The context provides the logic for one heterarchical grouping of objects and is more than a matter of convenience (Langford, 2012, p. 285).

The concept of abstraction and granularity posits that there exists an appropriate granularity to examine an SoI. The granularity is adjudicated by the chosen level of abstraction and context. While there is an appropriate granularity given abstraction and context, it is inevitable that many granularities exist as subjectively perceived by the various capability delivery organizations across a capability's life cycle.

It behooves the capability delivery organizations to be cognizant of the existence of potential incompatibility of reference abstraction and granularities that are all correct in their corresponding contexts. Without examining the system through the appropriate context, it is hard for capability delivery organizations to come to know of the full suite of emergent traits exhibited by the SoI beyond what they have designated as requirements for intended missions.

An increasing number of today's military capabilities are being achieved through a new system that became to be known as SoS (ODUSD [AT&L] SSE, 2008). The following section highlights the similarities and differences between a system and the SoS-class of system, and notes the implications for managing an SoS.

2. System of Systems

There are a number of definitions for SoS; Jamshidi (2009) reviewed upwards of six potential definitions before putting forward his own definition that "SoS are large-scale integrated systems that are heterogeneous and independently operable on their own, but are networked together for a common goal" (p. 2).

Maier (1998) argued that it was useful to distinguish SoS from various complex and large-scale systems, allowing the grouping of distinct demands to the design, development and operation of such a class of system. Five characteristics of SoS that made the design, development and operation of this taxonomical branch of system more challenging have often been attributed to Maier (1998), who in his 1998 paper, only considered the first two characteristics

to be key: (1) operational independence of component systems, (2) managerial independence of component systems, (3) emergent behavior, (4) geographical distribution, and (5) evolutionary development processes.

The *SEBoK* (Pyster & Olwell, 2013) stated that while there were no agreed upon definitions the following definition and its implication as quoted from the *SEBoK* has received substantial attention:

> An SoS is an integration of a finite number of constituent systems which are independent and operatable, and which are networked together for a period of time to achieve a higher goal." It should be noted that according to this definition, formation of an SoS is not necessarily a permanent phenomenon, but rather a matter of necessity for integrating and networking systems in a coordinated way for specific goals such as robustness, cost, efficiency, etc.

Langford (2012) tabulated factors that determined the systemness of a collection of objects (pp. 199–200). With respect to distinguishing a system from an SoS, it was said that the parts of an SoS predominantly show reversible properties and attributes when taken apart from the whole, whereas the parts of a system often exhibit irreversible properties and attributes when severed from the whole (Langford, 2012). This difference between systems and SoS accentuated SEBoK's definition that the SoS is not a permanent phenomenon, and that the parts must be able to revert to their original properties, traits and attributes to execute their independently operatable purposes.

As an SoS is a system, an SoS would satisfy the definition of a system adopted in this thesis; it would, however, be more useful for evaluating various process models that provide for capability delivery to develop a set of qualifying factors to help discern the SoS class of systems.

With regards to the Maier's list of SoS characteristics, this research is premised on the first two characteristics of operational and managerial independence of the whole and its parts are necessary qualifiers. The third characteristic of emergent behavior, while necessary, does not help in distinguishing an SoS from the more generic class of systems. An SoI that

exhibits the fourth and fifth characteristics of geographical distribution and evolutionary development processes, though, might suggest an SoS is not necessary as these were decisions made out of choice instead of necessity.

As such, geographical distribution of parts and the use of evolutionary development processes are useful factors but not as conclusive. The characteristic of emergent behavior has already been subsumed under the definition of a system. For the purpose of this thesis, an SoS must exhibit the following factors in addition to fulfilling the definition of a system as laid out in the previous section:

- Operational independence of the parts from the whole: The parts must be able to operate independently when severed from the whole according to its own set of customer-operator purposes (Maier, 1998).
- Managerial independence of the parts from the whole: The parts are separately acquired and integrated but continue to maintain their own operations independent of the whole (Maier, 1998).
- Property and attribute reversibility of the parts from the whole: The parts take on different properties and attributes for the duration of operations as a whole, but reverts when severed from the whole (Langford, 2012, pp. 199–200).

Based on these qualifying characteristics, an Aegis cruiser is part of an SoS when we examine the Aegis cruiser in the context of four-phased[2] ballistic missile defense; the heterarchical granularity of the SoI expands to include ground-based interceptors, sea-based radars, a suite of radars in the United Kingdom, Aleutian Islands, Greenland, and California (*Fact sheet: The ballistic missile defense system*, 2013). At this level of granularity, the Ballistic Missile Defense System exhibits all three qualifying factors of an SoS. The parts retain their operational and managerial independence; for example, the ground-based interceptors may be designated to take out other air-borne targets apart from ballistic missiles or the Aegis cruiser could be tasked to a search and destroy mission unrelated to the SoS-level ballistic missile defense mission.

[2] Ballistic missiles follow a four-phased trajectory path: boost, ascent, midcourse, and terminal.

These qualifying factors of an SoS mean there would be greater management issues due to potential tension between the SoS management entity and the constituent system entities. The following list captures some of the management issues of an SoS adapted from Osmundson, Huynh, & Langford (2007):

- Initial agreement: Initial agreement of SoS objectives by decision makers depends on the number of business entities involved. Top-down mandate of objectives would be possible if the whole SoS was under the purview of a single entity, which might not be the case.
- Planning: The planning for an SoS has to consider the matching of operations of constituent systems to external systems.
- Organizing: Establishment and monitoring of processes that interface the SoS with constituent systems.
- Directing and reporting: Clear, concise and complete communication channels must be established for the SoS and constituent systems. Metrics must be developed, collected and reported to the SoS-level.
- Design: Each constituent system has to balance the need to share classified or proprietary design information against the benefits of developing the SoS.
- Common interfaces: Interfaces must be identified and managed to ensure interoperability between constituent systems.
- Negative emergent behavior: The SoS may exhibit unexpected negative emergent behavior that is detrimental to the SoS and constituent systems.

As practitioners and academics better understood the concept of SoS through work experience and research, four types of SoS were identified based on the type of management and technical control the SoS-level has over its constituent systems (Dahmann et al., 2008):

- Directed SoS is one in which the integrated system of systems is built and managed to fulfill specific purposes. The Future Combat Systems is a directed SoS. It is centrally managed during long-term operation to continue to fulfill those purposes as well as any new ones the system owners might wish to address. The component systems maintain an ability to operate independently, but their

normal operational mode is subordinated to the central managed purpose.

- Acknowledged SoS has recognized objectives, a designated manager, and resources for the SoS; however, the constituent systems retain their independent ownership, objectives, funding, as well as development and sustainment approachesic Missile Defense System is an example of an Acknowledged SoS. The BallistChanges in the systems are based on collaboration between the SoS and the system.
- In Collaborative SoS, the component systems interact more or less voluntarily to fulfill agreed-upon central purposes. The Internet is a collaborative system. The Internet Engineering Task Force works out standards but has no power to enforce them. The central players collectively decide how to provide or deny service, thereby providing some means of enforcing and maintaining standards.
- Virtual SoS lacks a central management authority and a centrally agreed-upon purpose for the system of systems. The Global Information Grid is an example of a Virtual SoS. Large-scale behavior emerges—and may be desirable—but this type of SoS must rely upon relatively invisible mechanisms to maintain it.

Dahmann et al. (2008) asserted that the DoD has faced more capability delivery challenges from acknowledged SoS than the other three types. As an acknowledged SoS is not under the control of a single entity, it would face greater issues of initial agreement of SoS objectives with constituent system entities. Constituent systems might already be in development or even operation, adding complexity to the planning, organization, direction, reporting and design of the whole SoS (Osmundson et al., 2007). These constituent systems acknowledge the SoS capability objectives but are needed for their original requirements. The dual levels of management, objectives and funding create management challenges for both the acknowledged SoS and its constituent systems (Dahman et al., 2008).

It can be seen that the SoS-class of systems is subject to increased friction amongst constituent system entities and creates management challenges during capability delivery. These dynamic and competing behaviors of an SoS have to be captured in the CDS ontology.

C. EMERGENCE IN SYSTEMS

It is clear from the definitions of systems that a key reason for assembling a system would be for the emergent behavior that would result. Emergence is a condition when the whole is equal to the parts *plus* traits that are related to the context of the interaction of the parts (Langford, 2013a). In other words, through the interactions of the parts, the whole is able to achieve objectives greater than the sum of its parts. Jamshidi (2009) stated that the concept of the whole being more than the sum of its parts could be traced back to as early as Aristotle, but the utility of emergence beyond informed thinking continues to be questioned.

As cited in Jamshidi (2009), Holland pointed out that "emergent patterns are not adequately understood without the appreciation of the context within which the patterns exist" (p. 174). This is especially the case of SoS, where the context could be highly variable; "emergence has far-reaching implications for how we think, make decisions, and interpret results related to design, deployment, and transformation of SoS solutions" (Jamshidi, 2009, p. 174). Jamshidi, 2009 examined the nature of emergence through three perspectives: (1) philosophical; (2) axiomatic; and (3) methodological. The same approach to understanding emergence was adopted for the purpose of this thesis. The philosophical perspective deals with the commonly held worldviews on emergence. The axiomatic perspective examines the axiomatic principles that support a robust perspective for emergence in SoS. The methodological perspective deals with the general methodological considerations that could be adapted to specific contexts to account for emergence.

1. Philosophical Perspective

Jamshidi (2009) asserts the importance of understanding and appreciating the existence of varying world views on emergence, as there is greater potential for conflicts in SoS capability delivery organizations holding different worldviews.

These worldviews are reference frames through which we "give meaning to actions, decisions, and events as they unfold" (Jamshidi, 2009, p. 175). He

argues that the organizations' philosophical leanings in the epistemological and ontological frames shape how they perceive emergence. Figure 3 shows the philosophical spectrum of these two frames. The epistemological frame relates to how organizations' believe they perceive, collect, and communicate *knowledge*, while the ontological frame deals with the organizations' belief in what is *reality*.

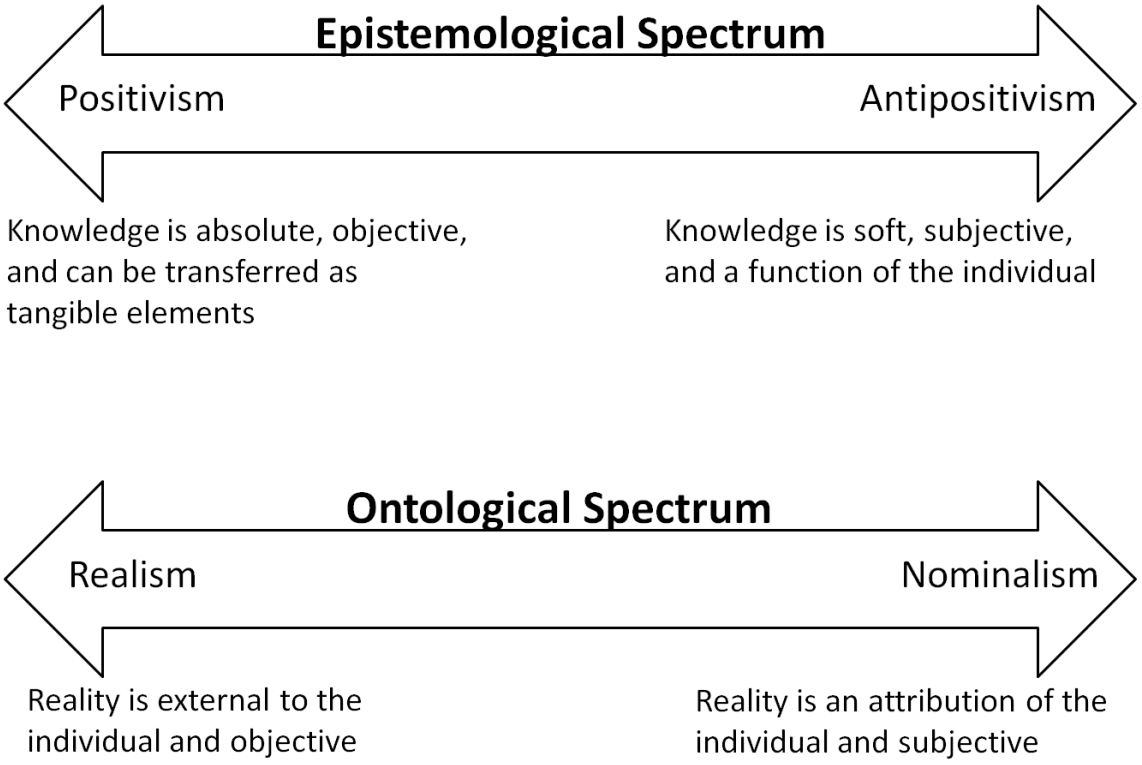


Figure 3. Philosophic-level Spectrum (After Keating, 2005).

A capability delivery organization leaning towards the positivism end of the spectrum would tend to take the stance that all system emergences can be predicted based on "absoluteness of system knowledge," while another organization with antipositivistic leanings would not expect absolute system knowledge and hence accept the existence of indeterminable emergence and its variety of interpretations (Jamshidi, 2009). Similarly, an organization that takes a realistic view might be inclined only to accept emergence as it is measured, while

a nominalistic organization would accept that the reality of emergence as subjective to the beholder (Jamshidi, 2009).

Emergence of a system in a given context could only be commonly discussed if the organizations examined the system at the same level of abstraction and granularity. As indicated earlier in the discussion on abstraction and granularity, the appropriate hierarchical and heterarchical view of the parts of a system would be adjudicated by the context in which the emergence of interest arises (Langford, personal communication, July 8, 2013). As such, capability delivery organizations having different abstractions and granules in mind would be debating the emergent properties, traits and attributes of a system that was only common in the name of the system.

It is not the purpose of this thesis to argue the philosophical merits or superiority of worldviews, but it could be surmised that capability delivery organizations would stand to gain if they recognize that other organizations may interpret emergence differently. These organizations should plan how to analyze, measure, and discuss emergence productively.

Kasser (2012) discussed two relevant orthogonal dimensions to a problem, the first being complexity and the second being complicatedness. On one hand, the complexity of a problem is an external objective characteristic "determined by the number of issues, functions, or variables involved in the problem; the degree of connectivity among those variables; the type of functional relationship among those properties; and the stability among the properties of the problem over time" (Kasser, 2012). On the other hand complicatedness is subjective to the level of competency held by the capability delivery organization with respect to the required domain expertise to examine the problem (Kasser, 2012).

An emergent trait that exhibited in a complex operational context might be too complicated for one capability organization, but is relatively easy for another organization with the knowledge and tools to measure it. Emergence that could

not be determined through analysis nor measured by contemporary methods and tools would still be determinable or measurable given the advancement in theory, measurements, and tools eventually (Langford, private communication, July 3, 2013).

The implications for the CDS ontology is that there exists a fog of emergence clouding the capability delivery organization's subjective perception of the objective manifestation of an SoI's full suite of emergent traits. This fog of emergence is modified by the organization's competency in the requisite engineering domain to determine or measure the emergent trait.

2. Axiomatic Perspective

The axiomatic perspective is a view comprised of knowledge that is regarded as established in the field. There is much development in the knowledge that is directly applicable to systems, but not much in the way of theories relating to emergence, despite emergence being considered axiomatic with regard to systems[3] (Jamshidi, 2009). The following section summarizes the explications on emergence derived from "systems-based concepts that are supportive of the emergence perspective" (Jamshidi, 2009).

1. Holism (Jamshidi, 2009) Skyttner "suggests that we cannot understand a complex system through reduction to the component or entity level" (p. 178). Holism is opposed to reductionism which believes that a complex system is simply the sum of its parts and hence could be absolutely analyzed at increasingly finder levels of details. Holism states that organizations have to analyze a system holistically in its context to fully comprehend associated emergent traits. Reductionist methods could still be used to study emergence if it could separate the parts from the whole, and "identify nonlinearities in performances and results to quantify losses" (Langford, 2012, p. 227).

2. Context "is the circumstances, factors, conditions, and patterns that both enable and constrain a complex system solution" (Keating,

[3] Jamshidi (2009) made the general statements with regard to emergence in particular to SoS, but as argued by this author, emergence is a characteristic of all systems and not exclusive to SoS.

2005). The context in which the system is used adjudicates the appropriate level of granularity and abstraction of the "whole" through which emergence could be known (Langford, 2012). These contexts are more often "as-deployed" than "as-intended," and hence it is a fundamental error to analyze a system solely based on designed intentions (Jamshidi, 2009, p. 180). As the context is external to the system and dependent on the military performers that use it, there could be a multiplicity of contexts and associated emergence beyond those envisioned by the organizations that designed and implemented it.

3. Complementarity "suggests that any two different perspectives … of a system will provide different knowledge of the system" (Jamshidi, 2009, p. 180). As a logical result of holism and multiplicity of contexts, every context in which the system is used while being potentially incompatible would still be valid and serve to complement the holistic impression of the system (Jamshidi, 2009).

4. System darkness "is a concept that recognizes there can never be complete knowledge of a system" according to Skyttner (as cited in Jamshidi, 2009, p. 181). Wolpert (2008) rigorously proved that an organization could never infer entirely correct knowledge of the system of which the organization is a part. This means that the knowledge of a system from an internal perspective is incomplete and speculative. Knowledge of a system and its associated emergence within the contexts in which it operates unfolds together with system operation and observations (Jamshidi, 2009).

5. Dynamic stability "holds that a system remains stable as long as it can continue to produce required performance during environmental turbulence and changing conditions" (Jamshidi, 2009, p. 182). Neither the system nor the context in which it operates remains the same, and so stability in the system is achieved through adjustments to disturbances in system performance (Jamshidi, 2009). Emergence is a result of the EMMI exchanged between objects of the system and context to achieve a natural stable state (Langford, 2012).

6. Metasystem "provides the structure of relationships that integrates the SoS[4]" according to Beer (as cited by Jamshidi, 2009, p. 181) and could be depicted as a three-dimensional coordinate system with one axis running the spectrum from: (1) integration to

[4] The five axioms relating to emergence were generally applicable to systems, but the axiom regarding metasystem is more pertinent to SoS, as a non-SoS would not be subjected to tension along the integration-autonomy axis.

> autonomy; (2) a second axis spanning stability to change; and (3) the third-axis of purposeful design to self- organization (Jamshidi, 2009). An SoS is subjected to formal structural relationships, but the “balance in tensions might shift through the life of the SoS” (Jamshidi, 2009, p. 181). It was said that a variety of emergence would be produced by the SoS to resolve structural tensions due to such shifts in balance along the axes of the metasystem. (Jamshidi, 2009).

From the axiomatic perspective, dynamic stability supports emergence as an intrinsic phenomena that results from EMMI interactions between parts of a system to perform a function. While the impact of emergence is greatest when unexpected, it is wrong to only associate emergence with surprise. The CDS ontology has to reflect this intrinsic manifestation of emergent traits regardless of whether it is known or not.

The concepts of holism, contexts, complementarity, and system darkness, adds to the fog of emergence in the CDS ontology. The full suite of emergent traits that manifest after system functions are performed is a complementary result of all the as-deployed mission contexts beyond those that were intended. System darkness posits that the capability delivery organization might not accurately infer the full suite emergent traits because of imperfect knowledge regarding the contexts. The fog of emergence in the CDS ontology has to incorporate this subjective knowledge of known missions against an all-omniscient objective list of as-deployed mission contexts. If a mission context is intended, the emergent traits that manifest in the intended context are determinable based on the capability delivery organization’s competencies in the requisite engineering domain. If the mission context is unknown, the emergent traits for that unknown context would be indeterminable, as even the most competent organizations would not be able to determine emergent traits without first knowing the context in which they manifest.

Finally, the concept of metasystem stresses implies that a comprehensive CDS ontology with emergence has to model the fluctuations in the emergent

traits manifested by the same SoS in the same mission context, due to perturbations along the three axes mentioned previously.

3. Methodological Perspective

Keating (2005) suggests that the philosophic perspective would inform the axiomatic perspective which in turn informs the methodological perspective. The methodological perspective is concerned with “guiding frameworks that are used to guide inquiry and gain knowledge regarding complex systems” (Keating, 2005).

Jamshidi (2009) observes that many systems engineering processes have been developed and applied successfully, but they are insufficient to be considered as a methodology. He opines that any combination of the following six conditions would favor the guiding hand offered by a systems-based methodology over prescriptive traditional systems engineering processes: (1) turbulent environmental conditions; (2) ill-defined problem conditions; (3) contextual dominance; (4) uncertainty for approaches; (5) ambiguous expectations and objectives; and (6) excessive complexity (Jamshidi, 2009). The attributes for systems-based methodologies are identified in Table 1.

Table 1. Attributes of a systems-based methodology (From Jamshidi, 2009, p. 179).

Methodology Attribute	Description
Transportable	Capable of application across a spectrum of complex systems engineering problems and contexts. The appropriateness (applicability) of the methodology to a range of circumstances and system problem types must be clearly established as the central characteristic of transportability.
Theoretical and philosophical grounding	Linkage of the methodology to a theoretical body of knowledge as well as philosophical underpinnings that form the basis for the methodology and its application.
Guide to action	The methodology must provide sufficient detail to frame appropriate actions and guide direction of efforts to implement the methodology. While not prescriptively defining "how" execution must be accomplished, the methodology must establish the high-level "whats" that must be performed.
Significance	The methodology must exhibit the "holistic" capacity to address multiple problem system domains, minimally including contextual, human, organizational, managerial, policy, technical, and political aspects of an SoS problem.
Consistency	Capable of providing replicability of approach and results interpretation based on deployment of the methodology in similar contexts. The methodology is transparent, clearly delineating the details of the approach for design, analysis, and transformation of the SoS.
Adaptable	Capable of flexing and modifying the approach configuration, execution or expectations based on changing conditions or circumstances – remaining within the framework of the guidance provided by methodology but adapting as necessary to facilitate systemic inquiry.
Neutrality	The methodology attempts to minimize and account for external influences in the application and interpretation. Provides sufficient transparency in approach, execution, and interpretation such that biases, assumptions, and limitations are capable of being made explicit and challenged within the methodology application.
Multiple utility	Supports a variety of applications with respect to complex SoS, including new system design, existing system transformation, and assessment of existing complex SoS initiatives. The methodology must provide for higher levels of inquiry and exploration of problematic situations, generating sufficient structuring and ordering necessary to move forward.
Rigorous	Capable of withstanding scrutiny with respect to (1) identified linkage/basis in a body of theory and knowledge, (2) sufficient depth to demonstrate detailed grounding in relationship to systemic underpinnings, including the systems engineering discipline, and (3) capable of providing transparent results that are replicable with respect to results achieved and accountability for explicit logic used to draw conclusions/interpretations.

The methodology perspective shall be used to assess the normative body of knowledge developed in this thesis based on the CDS ontology with emergence.

D. SYSTEM ENGINEERING LIFE CYCLE AND PROCESS MODELS

SE is not a new discipline but has been brought to the forefront when DoD acquisition policies mandated its use throughout a system's life cycle in 2006 (ODUSD [A&T] SSE, 2006). A number of SE process models have been developed over time that could be applied during a system's life cycle.

In this thesis, the parts and processes that comprise the CDS are taken to be the various capability delivery organizations that are responsible for the capability at the various life cycle phases using a particular SE process model to guide their interactions and work through these phases. The EMMI exchanges are the flow of intellectual properties, life cycle deliverables, and resources required for the various milestones and work packages. Hence, we began by taking a look at the generic system life cycle model that forms the common theme linking the variety of systems engineering process models and the acquisition system that form a key mechanism for the system's progress through its life cycle.

How SE process models alter the life cycle phases could be mapped, and as a result, codified as part of the capability delivery model based on the CDS ontology with emergence.

1. System Life cycle

Langford (2012) offers a nuanced distinction between a system's life cycle and the processes it goes through during its life (emphasis added) (p. 233–234):

> The systems life cycle perspective captures three issues: "(1) how comfortably the solution reflects life cycle needs; (2) the broader context in which the design is considered to have utility; and (3) the flexibility to incorporate cross-disciplinary views. …Life cycle can be seen as a structured progression from an initial beginning state to an end state, often thought of as from inception (beginning of life) to disposal (end of life). *Life cycle is not comprised of sequential or successive processes. Yet, life cycle discussions are appropriate to all processes and activities.* It is instructive to consider the life cycle of the problem, the stakeholder needs, the development effort, the product, and the product uses.

A distinction would be made for this thesis with regards to systems life cycle models and SE process models in order to examine the effects of nesting different SE processes within a system's life cycle.

This thesis would use the Defense Acquisition Management System (DAMS) as a working implementation of a generic systems life cycle. The DAMS answers "what needs to be done" to ensure standardization of terms of references, decision points, and of well-known deliverables across key stakeholders from an acquisition perspective (Office of Under Secretary of Defense Acquisition, Technology & Logistics [OUSD AT&L], 2008a). The various systems engineering process models answer "how to do it" and "for how long" to guide the capability delivery organizations from a systems engineering perspective.

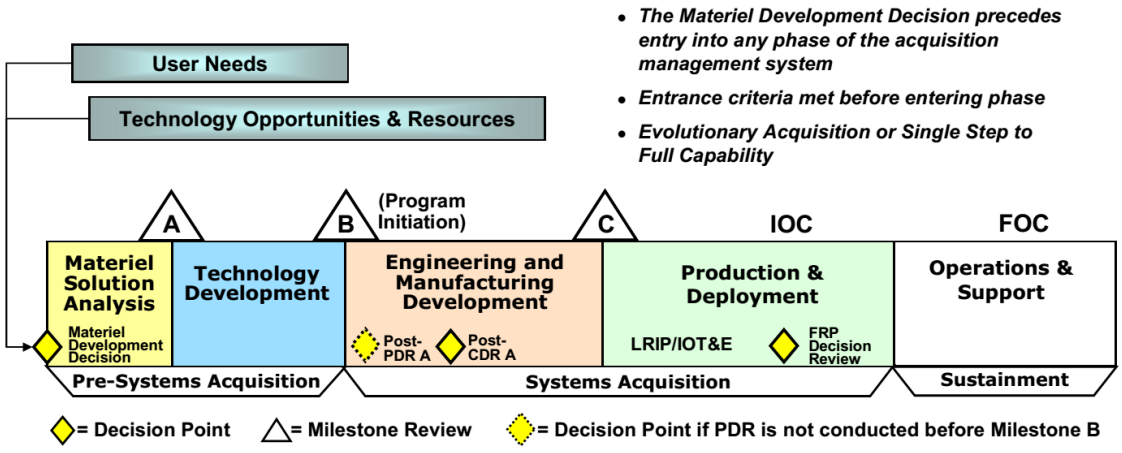


Figure 4. The Defense Acquisition Management System (From OUSD AT&L, 2008).

The DAMS is a working elaboration of the generic life cycle model from an acquisition perspective (OUSD AT&L, 2008). According to the Defense Acquisition University (DAU, 2008):

> The Materiel Development Decision (MDD) is the formal entry point into the acquisition process and is mandatory for all programs. It identifies a gap in capability and develops requirements to fill that

gap. The decision is documented in the Acquisition Decision Memorandum. The MDD consists of identification of a capability gap, a description of related risks, and a recommendation of whether or not to enter the acquisition process or use a non-materiel solution.

The MDD for a materiel solution has to precede entry into the acquisition process regardless of point of entry. The DAMS is comprised of five phases, three major milestones and clear regulatory deliverables and acquisition processes that had to be adhered to unless otherwise tailored by Milestone Decision Authorities (MDAs) as shown in Figure 4 (OUSD AT&L, 2008). They are: (1) the materiel solution analysis (MSA) phase; (2) the technology development (TD) phase; (3) the engineering & manufacturing development (EMD) phase; (4) the production & deployment (P&D) phase; and (5) the operations & support (O&S) phase.

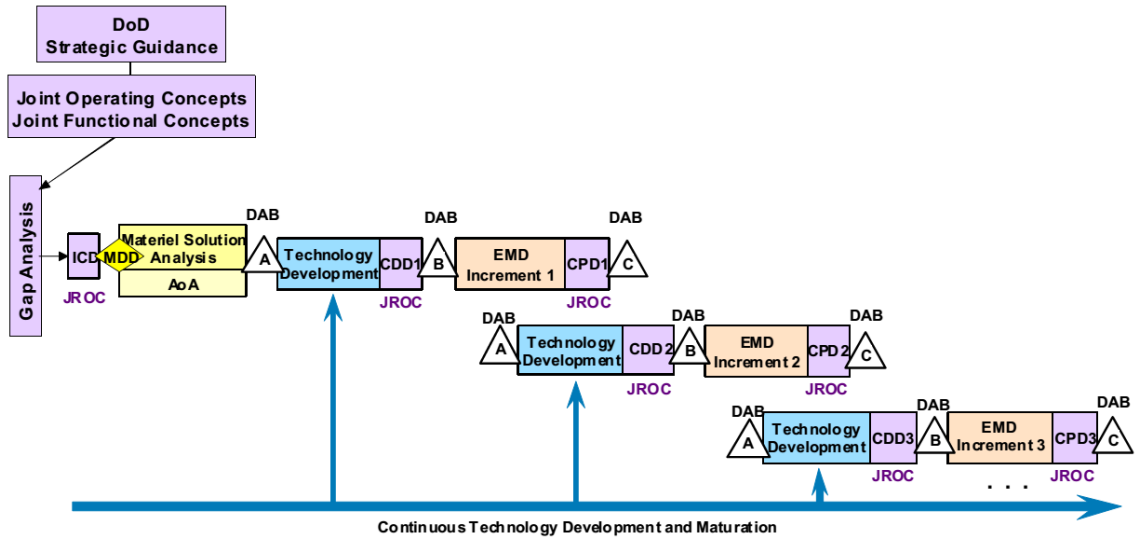


Figure 5. Requirements and Acquisition Flow of DAMS phases in Evolutionary Acquisition (From OUSD AT&L, 2008).

While the five phases could be sequential, they are not required to be executed in sequence. In fact, DoD's preferred acquisition strategy is "evolutionary acquisition," where a capability is delivered in increments with a recognition that capability needs may change and the capability improved in the future (OUSD AT&L, 2008). Figure 5 shows an example of how the requirements and acquisition flows through the DAMS phases for a capability with a disciplined approach to maturing technologies before development and its eventual production.

The following five tables (Tables 2–6) summarize the corresponding five phases of the Defense Acquisition Management System. Each table starts with the purpose of the phase, its pre-conditions, and ends with the post-conditions. The main body of the table describes the activities within the phase as well as important exceptions (if any). These activities are labeled with a prefix based on the abbreviations for their phase, followed by a running number that roughly indicates the order in which the activities occur. Most of the activities describe work packages to be done, and some describe important milestones and events. Those activities that are events have a character "e" appended to the end of their labels. For example, "MSA.4e" would denote the fourth activity of the MSA phase, and that it is an activity that denotes an event.

a. Materiel Solution Analysis Phase

Table 2. MSA Purpose & Description (From OUSD AT&L, 2008).

		Material Solution Analysis			
Purpose		Assess potential materiel solutions			
		Determine appropriate entry phase in DAMS after current phase			
Pre-conditions		Initial Capabilities Document: preliminary CONOPS, capability needs, operational risk, justification			
		Materiel Development Decision(MDD), Analysis of Alternatives (AoA) study guidance, Initial review date			
	S/N	**Activity**	**Product**	**Organization**	**Type**
Phase description	**MSA.1**	AoA study preparation to do preliminary assessment of materiel solutions, identify key technologies, and life cycle costs	AoA study Plan	Lead DoD Component	Program Elements
	MSA.2	AoA and Materiel Solution Analysis to develop Measures of Effectiveness (MoEs), cost, schedule, CONOPS, and risk of alternatives. Identify CTEs for each materiel solution and their tech-readiness, integration, and production risks.	AoA	Lead DoD Component	Program Elements
	MSA.3	Prepare appropriate DAMS Mile Stone (MS) artifacts	Depends on corresponding artifacts for MS A, B, or C	Lead DoD Component	Program Elements
	MSA.4e	Initial review of AoA and appropriate DAMS MS artifacts, and to decide if more reviews are needed.		MDA, Lead DoD Component	Event
Post-conditions		Completed AoA, Approved ICD, appropriate DAMS entry phase determined			

b. Technology Development Phase

Table 3. TD Purpose & Description (From OUSD AT&L, 2008).

		Technology Development Phase			
		Reduce technology risk			
		Determine and mature sets of technologies (CTEs)			
Purpose		Demonstrate CTEs on prototypes			
Pre-conditions		Completed AoA, proposed materiel solution, prepared MS A artifacts, full funding for TDP			
	S/N	**Activity**	**Product**	**Organization**	**Type**
Phase description	**pre-TDP.1**	Draft Technology Development Strategy (TDS): single-step or evolutionary acquisition, their schedules, cost, performance goals, exit criteria, and its increments and number of prototypes to be developed in this phase	MS A Artifact: TDS	Lead DoD Component	Program Elements
	pre-TDP.2	Estimate cost for each AoA solutions	MS A Artifact: Cost Estimate	Lead DoD Component	Program Elements
	TDP.1e	MS A: Review of proposed materiel solution, & MS A artifacts		Lead DoD Component, MDA	Event
	TDP.2	Preparation for Requests for Proposals (RFPs) after MS A approval	RFPs	PM	Program Elements
	TDP.3	Production of 2 or more prototypes	System components (prototype)	S&T communities	Program Elements
	TDP.4e	Prototype demonstrations		PM, S&T communities	Event
	TDP.5	Review life cycle costs based on demos	Life cycle Sustainment Plan (LCSP)s	PM	Program Elements
	TDP.6	Prepare Systems Engineering Plan (SEP) that includes Reliability, Availability, and Maintainability (RAM) strategy and reliability growth program for design and development.	SEPs	PM, Program Executive Officer (PEO)	Program Elements
	TDP.7e	Program Support Reviews (PSR): Review SEP & LCSP		Lead DoD Component	Event

	TDP.8	Prepare Preliminary Design Review (PDR) design artifacts: Candidate designs to establish allocated baseline (hw, sw, human), underlying architectures, and high-confidence design.	Preliminary designs	Lead DoD Component, S&T communities	Program Elements
	TDP.9e	Perform PDR: Inform requirement trades; improve cost estimation; identify System-level design, integration, & manufacturing risks.		User, Certification Authority, Lead DoD Component	Event
	TDP.10	Prepare MS B artifacts: Capability Development Document (CDD) to support initiation of acquisition program/increment, refine integrated architecture, and clarify how program would lead to war fighting capability. Includes detailed operational performance parameters.	CDD	User, PEO, PM, and [MDA]	Program Elements
	TDP.11e	CDD Approval		JROC	Event
Exception	**ex-TDP.1e**	If cost estimation increase by 25% over MS A certification, PM has to notify MDA for possible rescindment of MS A approval		PM, MDA	Event
	ex-TDP.2	If evolutionary, an MDA approved TDS is required for every increment with a MS A		Lead DoD Component, MDA	Program Elements
Post-conditions		Affordable program/increment of militarily useful capability has been identified			
		Technology and manufacturing processes for program/increment assessed and demonstrated in relevant environment			
		Manufacturing risks identified			
		Program/increment can be developed for production within 5 years			

c. Engineering and Manufacturing Development Phase

Table 4. EMD Purpose & Description (From OUSD AT&L, 2008).

		Engineering Manufacturing & Development			
Purpose		Develop a system or an increment of a capability			
		Complete full system integration			
		Develop an affordable and executable manufacturing process			
		Ensure & demonstrate -ilities & Human Systems Integration (HSI)			
Pre-conditions		CDD with KPPs, technology maturity of materiel solution, approved requirements, and full funding.			
		System concept selected, requirements approved, and PM assigned			
		Activity	**Product**	**Organization**	**Type**
Phase description	**pre-EMD.1**	Capability Development Document (CDD) to support initiation of acquisition program/increment, refine integrated architecture, and clarify how program would lead to war fighting capability. Includes detailed operational performance parameters	MS B Artifact: CDD	User, PEO, PM, and [MDA]	Work packages
	pre-EMD.2	Low-Rate Initial Production (LRIP) quantities (one unit to 10% of total), staffing estimates, business case, acquisition program baseline (APB)	MS B Artifacts: LRIP, staffing estimate, business case, APB	PM	Work packages
	EMD.1e	MS B: Review of MS B artifacts and initiation of acquisition program		PM, MDA	Event
	EMD.2	Preparation for final RFPs after MS B approval; specifically worded to only award to proposals based on CTEs that have been demonstrated in a relevant environment & offerors to specify technology readiness levels of CTEs	Final RFPs	PM	Work packages
	EMD.3	Preparation for Requests for Proposals (RFPs) for TDP after MS A approval	RFPs	PM	Work packages
Integrated System Design (ISD)	**EMD.4**	If no PDR prior MS B, PM to plan for PDR design artifacts: Candidate designs to establish allocated baseline (hardware, software, human system integration), underlying architectures, and high-confidence design.	Preliminary design (Architecture, component design, production baseline)	PM, S&T Communities	Work packages

		Engineering Manufacturing & Development			
	EMD.5e	If no PDR prior MS B, conduct Preliminary Design Review (PDR): Inform requirement trades; improve cost estimation; identify System-level design, integration, & manufacturing risks.	PDR report	PM, S&T Communities	Event
	EMD.6e	If PDR conducted within EM&D phase, conduct Post-PDR: Formal assessment where MDA considers PM's assessment and PDR report	Post-PDR assessment in Acquisition Decision Memorandum	PM, MDA	Event
	EMD.7	Prepare for critical design review (CDR)	Critical design (Architecture, component design, production baseline for all configuration item)	PM, S&T Communities	Work packages
	EMD.8e	Conduct CDR	CDR Report	PM, Subject Matter Experts (SMEs), CDR Chair	Event
	EMD.9e	Post CDR Review	Initial product baseline	PM, MDA	Event
System Capability & Manufacturing Process Demonstration	**EMD.10**	Prepare for System Capability & Manufacturing Process Demonstration	System component (including manufacturing processes)	S&T communities	Work packages
	EMD.11e	Repeated developmental test & evaluation (DT&E) of technical progress, operational assessments, use of M&S to demonstrate integration		PM, S&T communities, operational users	Event
	EMD.12	Prepare MS C artifacts	MS C artifacts including Capability Production Document (CPD)	PM, S&T communities	Work packages
	EMD.13e	CPD approval		JROC	Event
Post-conditions		Purpose achieved			

d. Production and Deployment Phase

Table 5. P&D Purpose & Description (From OUSD AT&L, 2008).

		Production & Deployment Phase			
Purpose		Achieve operational capability that addresses mission needs, established through operational test & evaluation (OT&E)			
Pre-conditions		Acceptable performance in DT&E, mature software capability, no significant manufacturing risks, manufacturing processes under control, approved ICD (if MS C is program initiation), approved capability production document (CPD), refined integrated architecture, -ilities, phased for rapid acquisition and fully funded			
		Activity	**Product**	**Organization**	**Type**
Phase description	**P&D.1e**	MS C: Authorize entry into LRIP; production/procurement for non-LRIP; limited deployment for software intensive systems		MDA	Event
	P&D.2	Complete manufacturing development for initial OT&E and to establish production base	Production base	S&T communities	Work packages
	P&D.3	Execute LRIP	Production-representative articles	S&T communities	Work packages
	P&D.4e	Perform IOT&E to rectify deficiencies in both articles and production base		S&T communities, operational users	Event
	P&D.5	Prepare for FRP (demonstrate control of manufacturing process and acceptable reliability, collection of statistical process control data, demonstrated control and capability of critical processes)	Beyond LRIP Report	PM, S&T communities	Work packages
	P&D.6e	Full-Rate Production (FRP) Decision Review	FRP decision in Acquisition Decision Memorandum	MDA, Congress, USD (AT&L)	Event
	P&D.7	Execute FRP and Deployment: Deliver system and materiel to users.	Initial Operational Capability System Components	S&T communities, operational users	Work packages
	P&D.8e	Perform Follow-on OT&E (FOT&E) to assess system performance	[New capability requirements]	S&T communities, operational users	Event
	P&D.9	Ensure military equipment valuation		PM, S&T communities	Work packages
Post-conditions		Initial Operational Capability (IOC) achieved			

e. Operations and Support Phase

Table 6. O&S Purpose & Description (From OUSD AT&L, 2008).

		Operations & Support			
Purpose		Execute support system that meet materiel readiness and O&S performance in a cost-effective manner over system's life cycle			
Pre-conditions		Acceptable performance in DT&E, mature software capability, no significant manufacturing risks, manufacturing processes under control, approved ICD (if MS C is program initiation), approved capability production document (CPD), refined integrated architecture, -ilities, phased for rapid acquisition and fully funded			
		Activity	**Product**	**Organization**	**Type**
Phase description	**O&S.1**	Life cycle sustainment: Continual engineering for RAM, HSI, environment, safety, occupational health, supportability and interoperability		PM, S&T communities, operational users	Work packages
	O&S.2e	Iterative reviews		PM, operational users	Event
	O&S.3	Prepare for disposal		PM	Work packages
	O&S.4e	Disposal		PM, S&T communities, operational users	Event
	O&S.5e	PEO annual review		PEO, PM	Event

2. Systems Engineering Process Models

A quote by Nogueira, Jones, & Luqi (2000), made in the context of software engineering, mirrors the development of SE as a discipline to find the right balance between order and chaos:

> The edge of chaos is defined as "a natural state between order and chaos, a grand compromise between structure and surprise" (Kauffman as cited in Nogueira et al., 2000). The edge of chaos can be visualized as an unstable partially structured state of the universe. It is unstable because it is constantly attracted to the chaos or to the absolute order.
>
> We have the tendency to think that the order is the ideal state of nature. This could be a mistake. Research … supports the theory that operation away from equilibrium generates creativity, self-organization processes and increasing returns (Roos as cited

> in Nogueira et al., 2000). Absolute order means the absence of variability, which could be an advantage under unpredictable environments.
>
> Change occurs when there is some structure so that the change can be organized, but not so rigid that it cannot occur. Too much chaos, on the other hand, can make impossible the coordination and coherence. Lack of structure does not always mean disorder.

When the use of SE throughout a system's life cycle was mandated in 2006 by acquisition policies, it was believed that SE would provide "an overarching process that the program team applies to transition from a stated capability need to an affordable, operationally effective and suitable system" (ODUSD [A&T] SSE, 2006). The speed with which the pendulum swung toward greater order was further accelerated by DoD's increasing reliance on the more complex SoS to implement user capabilities. As a result the *SE Guide for SoS* was introduced (ODUSD [A&T] SSE, 2008) as "the SE community has recognized the need for discipline and structure in the engineering of SoS" (Dahmann et al., 2008). However, the recent additions of agile methods to the *International Council on Systems Engineering (INCOSE) SE Handbook* (2010) indicate that the pendulum swing might have been reversed, with focus on increased agility based on potentially chaotic interpersonal emergent processes rather than being driven by the false comforts of an ordered plan.

Each of these revisions to DoD policies and additions to SE as a discipline have been accompanied by more SE processes; such as the classic single-pass Waterfall and Vee, the iterative and concurrent Dual-Vee, the evolutionary Spiral and, the latest Agile processes to list a few more prominent ones.

A process is defined by the *SEBoK* (Pyster & Olwell, 2013) "as a series of actions or steps taken in order to achieve a particular end; as a verb it is the performing of the operations. Processes can be performed by humans or machines transforming inputs into outputs." Langford (2012) succinctly defines

process models “as models that describe the stages in which the project team focuses on various milestones and deliveries.” The process model signifies what stage is next and what events constitute that stage.

According to the *SEBoK* (Pyster & Olwell, 2013), there are three categories of SE processes: (1) pre-specified and sequential processes (e.g., single-pass classic waterfall model), (2) evolutionary and concurrent processes (e.g., various forms of the Vee-model, spiral models, and waterfall with feedback) and (3) interpersonal and emergent processes (e.g., agile development, scrum and extreme programming). A list of SE process model adjectives together with their ontological relations to other adjectives is shown in Table 7.

Table 7. List of common SE process model adjectives.

Adjective	Description
Pre-specified	Describes a process whereby the system requirements were predetermined and fixed for the scope and life cycle of the system (Pyster & Olwell, 2013). It is in contrast to an evolutionary process.
Evolutionary	Describes a process where successive versions of a system are produced in response to discoveries surfaced by earlier versions and changing requirements (Forsberg, Mooz, & Cotterman as cited by Pyster & Olwell, 2013). It is in contrast to a pre-specified process.
Single-pass[5]	Describes a process where a complete system is produced during the first iteration of the process (Pyster & Olwell, 2013). It is in contrast to a multi-pass process.
Multi-pass	Describes a process whereby either the whole or a subset of the process model is repeated during the system’s life cycle (adapted from Pyster & Olwell, 2013). It is in contrast to a single-pass process.
Iterative	Describes a process whereby either the whole or a subset of the process model is repeated during the system’s life cycle (adapted from Pyster & Olwell, 2013). It is synonymous to multi-pass and is in contrast to a single-pass.
Incremental	An incremental process is an iterative process with the additional condition that the system requirements were contiguously partitioned and delivered in successive versions of increments in features and functions (adapted from Mooz, Forsberg, & Cotterman as cited by Pyster & Olwell, 2013).
Sequential	Describes a process where versions of a system is *defined and developed* strictly in sequence one after another (Pyster & Olwell, 2013). Note that the adjective is used to describe the sequential nature of the pre-production phases of the process and does not imply that the whole process has to be sequential. A sequential process model has

[5] Pyster and Olwell (2013) use the words “Single-step” and “Multi-step” to distinguish between a process that takes only a single pass to produce a complete system and another process that takes multiple pass to produce a complete system. The word “step” might be confounded with the steps that comprise a process model. As such, the word “pass” is used.

Adjective	Description
	the lowest overlap (in fact, no overlap) of pre-production phases when compared to opportunistic and concurrent process models.
Opportunistic	Describes a process where subsequent versions of a system after the first is *defined and developed* contingent on the presentation of a sufficiently attractive opportunity, such as maturing desired technology or availability of key personnel (Pyster & Olwell, 2013). An opportunistic process model has higher overlap of pre-production phases compared to a sequential process model, but less overlap compared to a concurrent one.
Concurrent	Describes a process where subsequent versions of a system after the first is defined and developed concurrently. While not necessary it is recommended to ensure concurrently produced versions of the system are contiguous parts of the system with low modular coupling (adapted from Pyster & Olwell, 2013). A concurrent process model has the highest overlap of pre-production phases compared to sequential and opportunistic process models.
Unconstrained	Describes a process where a system is produced through an unconstrained order of phases.
Ordered	Describes a process where a system is produced through a well defined order of phases.

In the following subsections, these seven process models are examined in detail: (1) Waterfall; (2) Waterfall-with-feedback; (3) Vee; (4) Evolutionary and Incremental Vee; (5) Spiral; (6) Agile; (7) and Wave. By understanding the individual characteristics and principles behind each process model, codified strategies for each SE process model are developed to guide the customized allocation of DAMS phases to be put through the CDS model. These SE process model strategies affect the execution of work packages and milestones through the DAMS phases and evolve the capability expressed as an instance of the CDS ontology in a different manner.

a. Waterfall

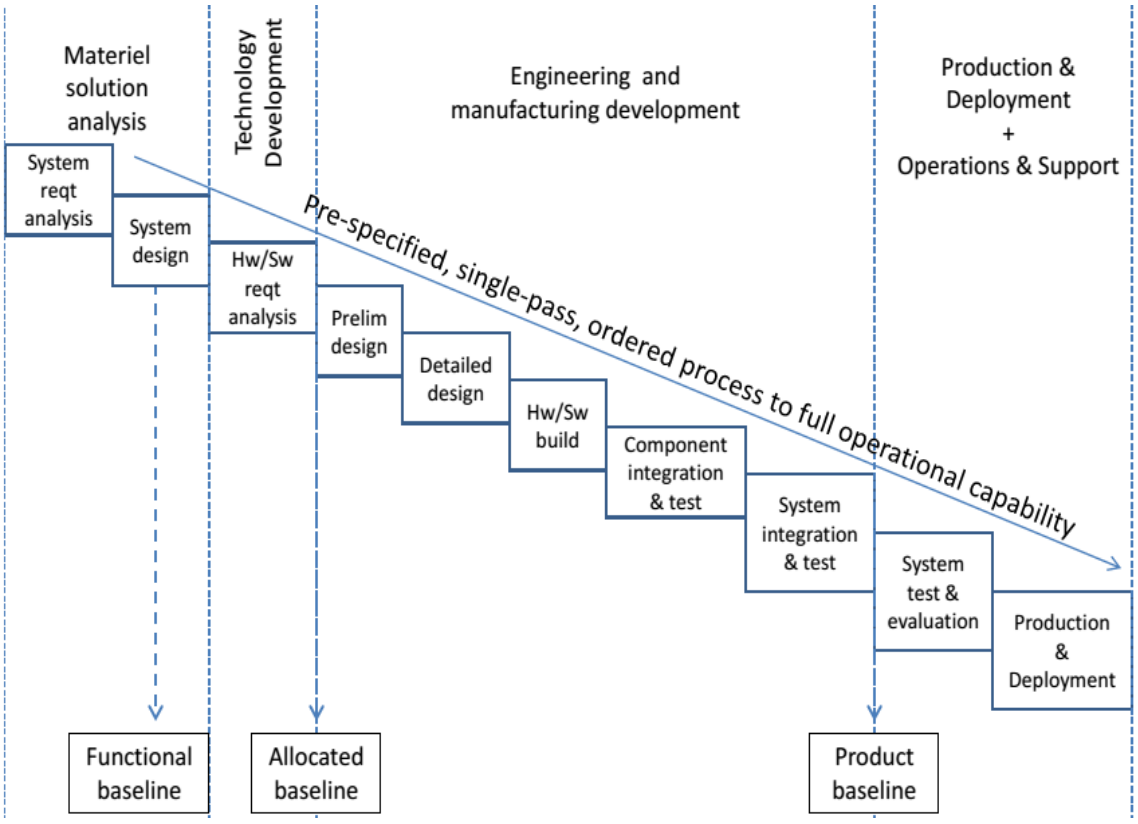


Figure 6. Waterfall SE process model (After USDoD, 1988).

The Waterfall is a pre-specified, single-pass, sequential and ordered SE process model, also known as the traditional waterfall model. For the rest of this thesis, we use the term “Waterfall” to mean this traditional version.

The Waterfall process model could be visualized as an ordered flow of steps, overlaid on the DAMS life cycle phases as shown in Figure 6. The key characteristic of the Waterfall process model is that it follows a strict progression through the life cycle stages without revisiting earlier steps (Pressman, 2010).

It is best used when the requirements for a problem are well understood in a context that is stable, therefore making it possible to capture all requirements and complete analysis before design starts. However, in the

modern context characterized by increasing system complexity and shifting context, the Waterfall's rigidity limits its application.

Modern systems are rarely implemented in a single-pass ordered flow, making it difficult to identify and freeze requirements at the start of programs. Even if requirements could be identified and frozen early, they could be invalid or irrelevant by deployment if the process execution takes a long time (Center for Technology in Government, 1998).

For the purpose of this thesis, a purist perspective was adopted and therefore, the Waterfall process model was treated as one that is totally insulated from changing requirements once the functional baseline has been established. Rework encountered would be due to errors in design or implementation that is only discovered during testing based on the set frozen requirements (Langford, private conversation, July 15, 2013).

b. Waterfall with Feedback

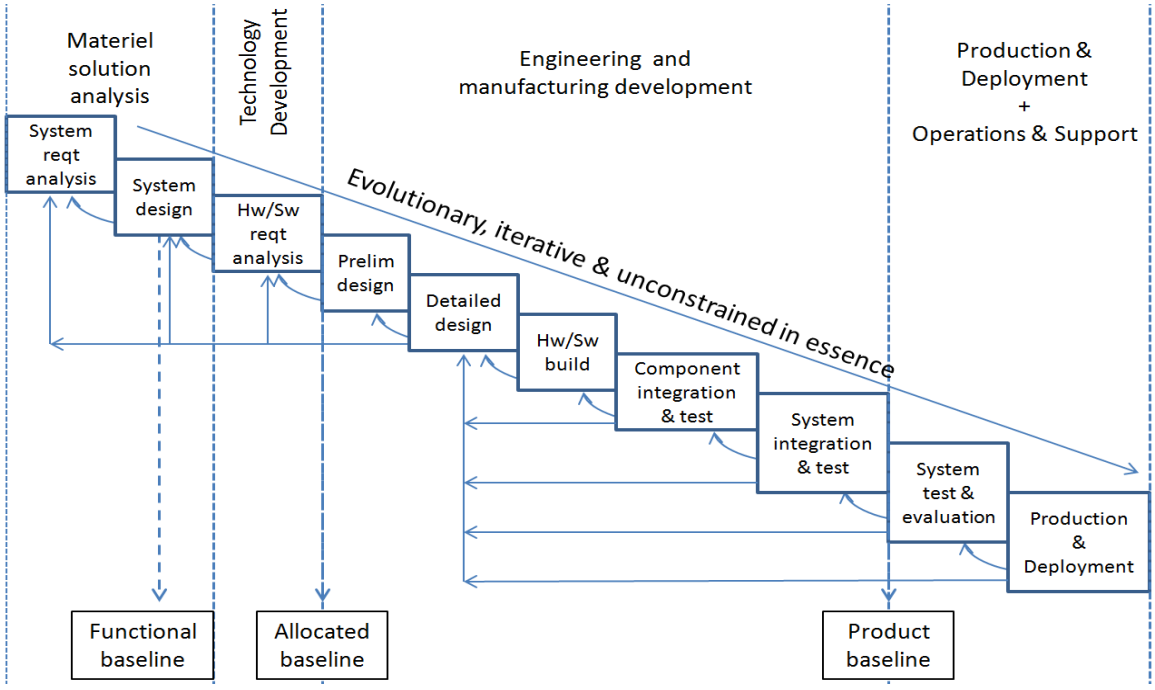


Figure 7. Waterfall with feedback (After Royce, 1970 and USDoD, 1988).

The classic Waterfall process model is often mistakenly attributed to Royce (1970), but Royce was not a proponent of the classic Waterfall. His 1970 paper was actually about a Waterfall-with-feedback model and criticized the use of the classic waterfall. The waterfall-with-feedback adds the feedback arrows as shown in Figure 7. It is a marked deviation from the Waterfall process model and is, in essence, as interpreted based on contemporary SE process terminology, an evolutionary, iterative, sequential, and unconstrained process model (Royce, 1970).

The model shows that at any part along the waterfall, the capability delivery organization could revisit an earlier step to rectify any unforeseen insufficiencies. Royce (1970) anticipated that unforeseen difficulties encountered after design might be so disruptive that the design has to be revisited bypassing the immediate steps preceding it. Likewise, the design change could be so drastic that it warrants a revisiting of the requirement steps (Royce, 1970).

The Waterfall-with-feedback process model has five guiding principles that further address the weaknesses of the classical Waterfall (Royce, 1970):

- Design first: The departure from the cascading ordered steps of the classic Waterfall shows up in the first guiding principle to start with design. To be more specific, a preliminary design is done together with system conceptualization and requirements analysis (Royce, 1970). The intent was to ensure that conceptualization and analysis were performed with a clearer appreciation of the consequences (Royce, 1970).
- Document the design: Focus on a disciplined approach to produce documents at every step (Langford, 2013b and Royce, 1970). Royce (1970) justifies the emphasis on design documentation as a tangible mean to track design progress, establish requirements traceability and is a key document referred to by downstream steps.
- Do it twice: If the system is an original concept, arrange it so that the system is only delivered on the second iteration of the whole waterfall-with-feedback process (Royce, 1970). The first iteration allows for experimentation to produce a prototype whose usage would provide feedback to all subsequent steps of the second iteration (Royce, 1970).

- Plan, control and monitor testing: Prioritize testing related activities, as formal testing occurs late in the process and consumes program resources (Royce, 1970). Capability delivery organizations shall build test models to uncover problems even before the formal test phase (Royce, 1970). The focus on ensuring correctness of implementation according to specifications is reflected in the updated waterfall models as embedded verification activities before the test phase (Langford, 2013b).
- Involve the user: Involve the user as early as possible and minimally during (1) systems requirements, (2) preliminary design review, (3) critical design review, and lastly (4) final system acceptance review (Royce, 1970). The frequent user involvement provides a means to evolve the requirements and system design continually and as early as practicable to avoid propagating invalid requirements and incorrect design/ implementation.

c. Vee

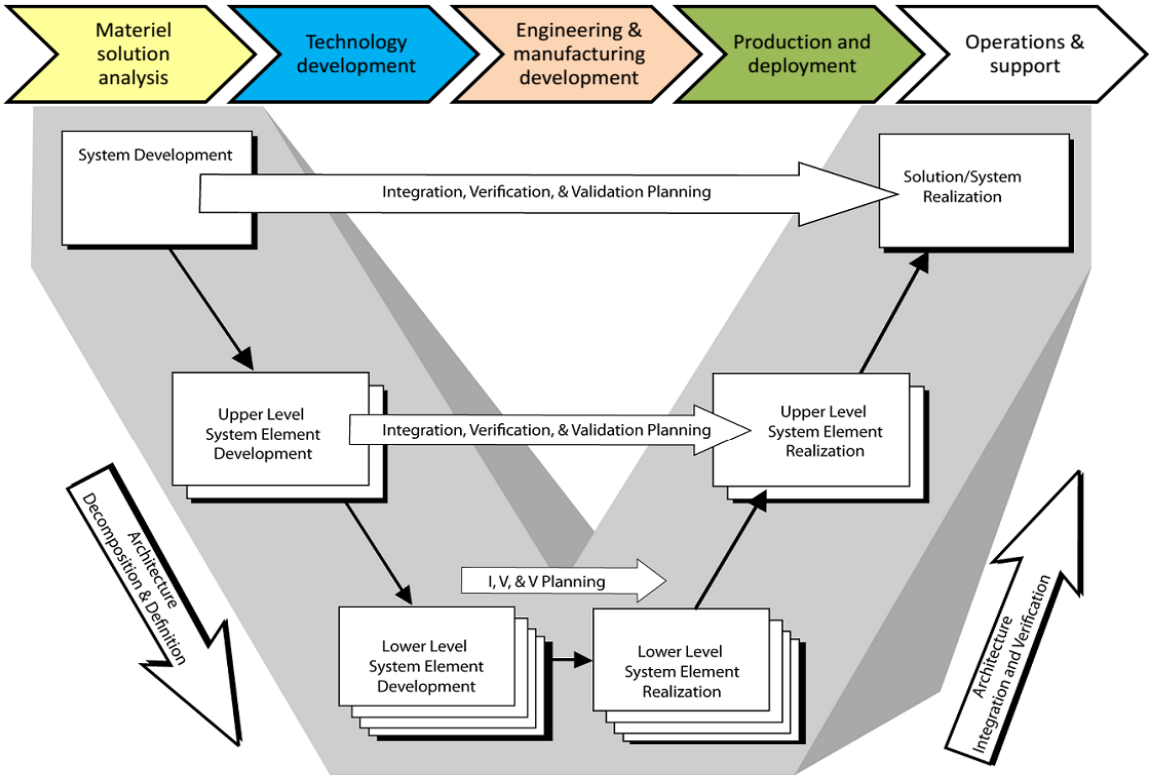


Figure 8. Vee model (After INCOSE, 2010).

The original Vee model introduced by Forsberg and Mooz in 1991 is a pre-specified, single-pass, sequential, and ordered process like the Waterfall model, but with three differences: (1) a greater focus on systems engineering activities; (2) continual need to verify and validate; and (3) evolving baselines of the system that is decomposed and defined as it moves down the left of the Vee and integrated and verified up the right of the Vee as shown in Figure 8 (Pyster & Olwell, 2013; and INCOSE, 2010). Kasser (2010), states that the Vee is a rearranged waterfall view “for use as a management tool showing the relationship between design activities and test activities” (Forsberg and Mooz, as cited in Kasser, 2010).

The Vee model corresponds to the DAMS phases, as shown in Figure 9, starting with solution-agnostic system conceptualization during the materiel solution analysis phase before moving down the left-Vee, to demonstrate and validate system concepts in the technology development phase. Engineering and manufacturing development occurs at the base of the Vee. Production and deployment take place along the right-Vee, and end with operations and support at the tip of the right-Vee.

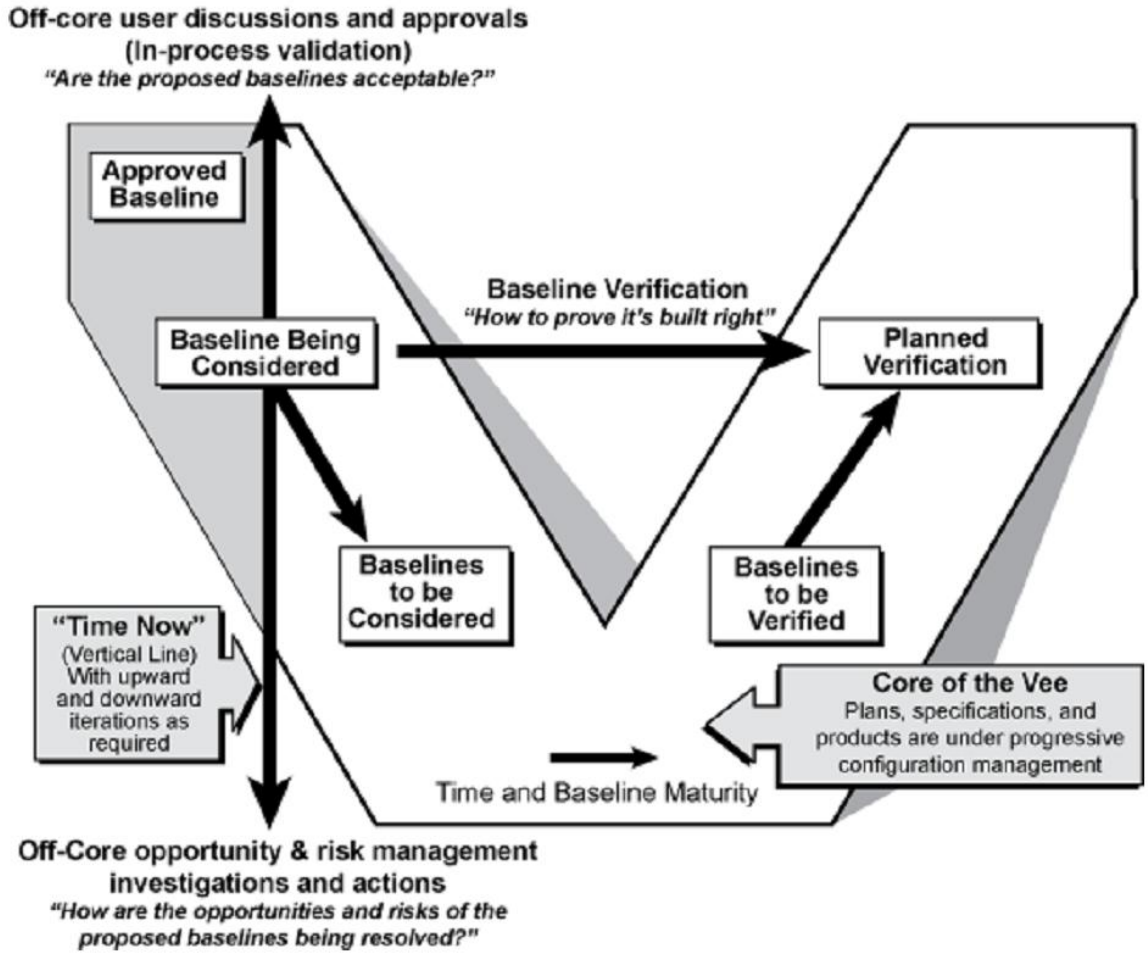


Figure 9. Left side of the sequential Vee (From INCOSE, 2010).

The Vee model is able to show an extremely high level snapshot of the system's passage of time, maturity, baselines as well as upward vertical validation and downward vertical investigations. According to the INCOSE (2010):

> In the Vee model, time and system maturity proceed from left to right. The core of the Vee (i.e., those products that have been placed under configuration control) depicts the evolving baseline from user requirements agreement to identification of a system concept to definition of elements that will comprise the final system. With time moving to the right and with the system maturity shown vertically, the evolving baseline defines the left side of the core of the Vee, as shown in the shaded portion of Figure 3–5 (Figure 10 in this thesis).

As entities are constructed, verified and integrated, the right side of the core of the Vee is executed (as shown in Figure 11). Since one can never go backward in time, all iterations in the Vee are performed on the vertical “time now” line. Upward iterations involve the stakeholders and are the in - process validation activities that ensure that the proposed baselines are acceptable. The downward vertical iterations are the essential off - core opportunity and risk management investigations and actions. In each stage of the system life cycle, the SE processes iterate to ensure that a concept or design is feasible and that the stakeholders remain supportive of the solution as it evolves.

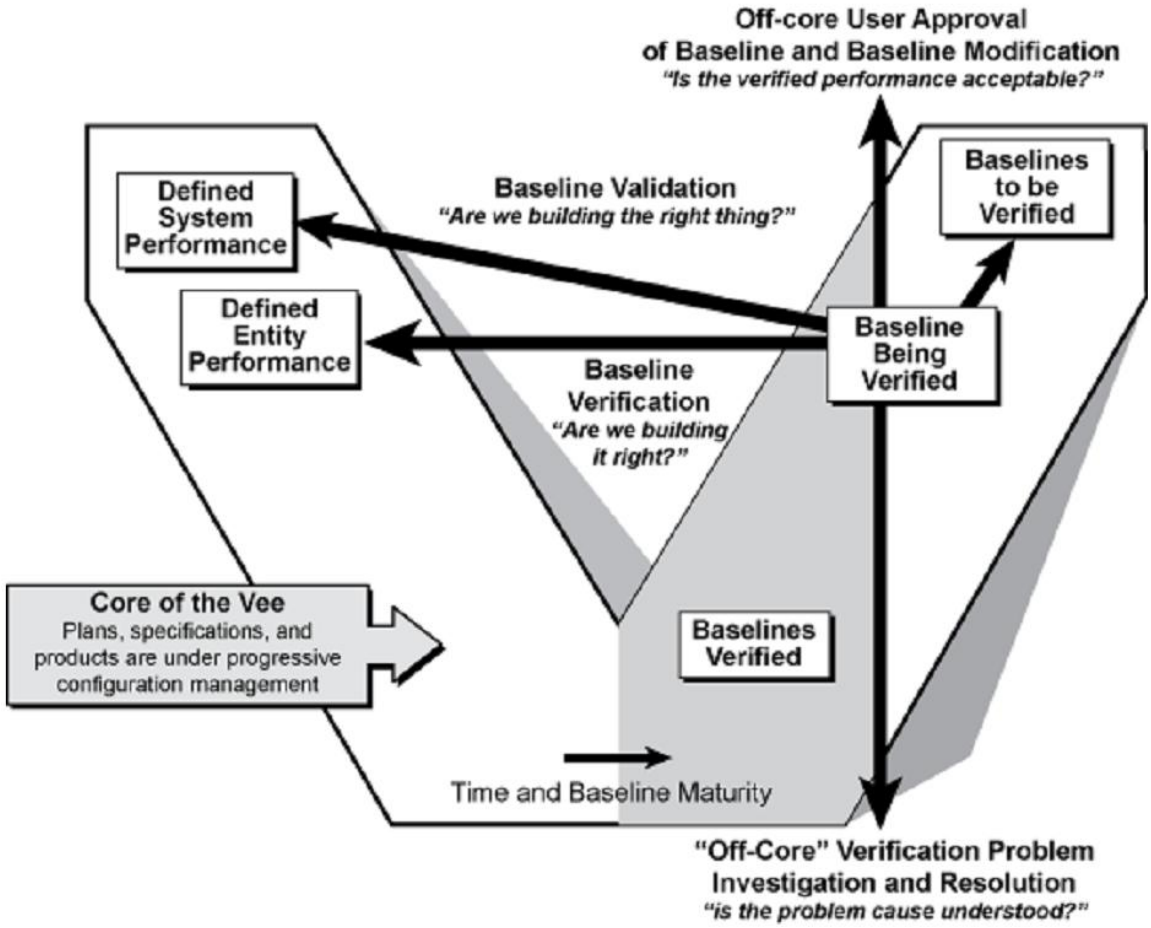


Figure 10. Right side of the sequential Vee (From INCOSE, 2010).

A problem with the Vee-model is that “practitioners tend to forget, or are unaware, that the Vee is a three-dimensional view as shown in Figure 11 and in its two-dimensional representation it is only an overview of some of the

aspects of the project cycle relating to the development to test and evaluation at the various phases of the system life cycle while abstracting out all other information" (Kasser, 2010). A third dimension of the Systems Analysis and Design Process has to be applied when going down the left-Vee and when coming up on the right-Vee a System Verification and Integration Process has to be applied (Forsberg & Mooz, 1995).

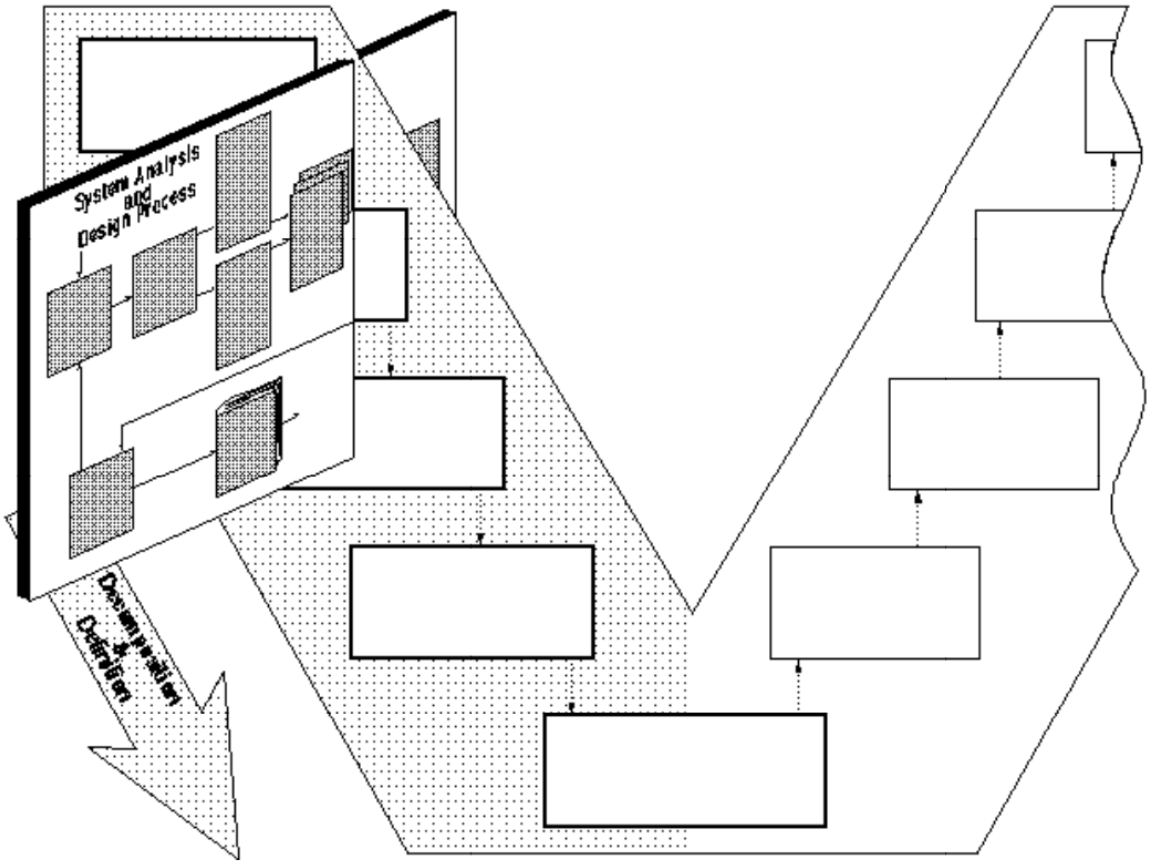


Figure 11. Application of System Analysis and Design Process to the Concept Exploration phase (From Forsberg & Mooz, 1995).

The Vee-model, being a derivative of the classic Waterfall, suffers from the same weakness of being unable to consider changes in customer needs during development of the solution system (Kasser, 2010).

d. Evolutionary and Incremental Vee

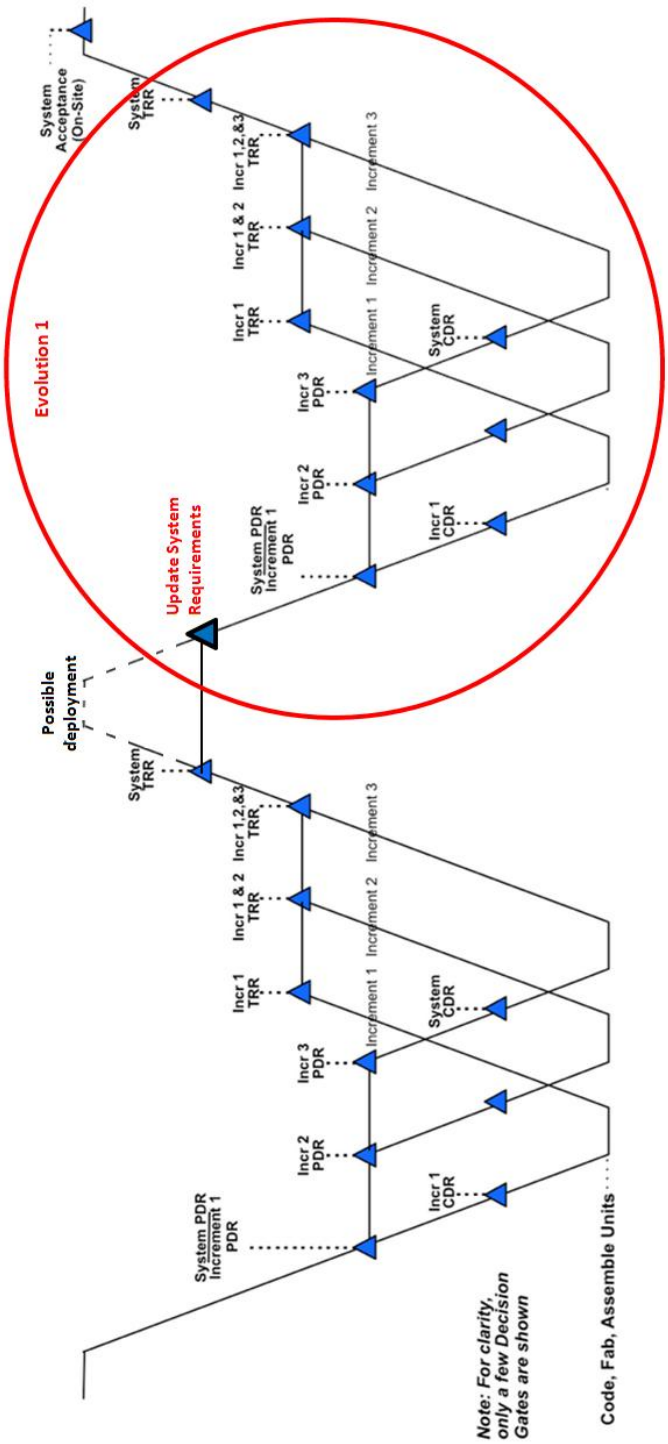


Figure 12. Evolutionary and Iterative Vee Model (After Forsberg & Mooz, as cited in Pyster & Olwell, 2013).

The Evolutionary and Iterative Vee is, as the name implies, is an updated variant of the Vee model with evolutionary and iterative features as shown in Figure 12. According to Pyster & Olwell (2013), it is used when:

> (1) rapid exploration and implementation of part of the system is desired; (2) the requirements are unclear from the beginning; (3) funding is constrained; (4) the customer wishes to hold the System of Interest open to the possibility of inserting new technology at a later time; or (5) experimentation is required to develop successive prototype versions.

The Evolutionary and Iterative Vee differs from the Vee model in that it need not be plan-driven and increments could be opportunistic in nature, contingent on maturing technology or changes in needs or requirements (Pyster & Olwell, 2013). As shown in Figure 12 the capability delivery organizations use increments to develop parts of the system first, but the system cannot function as a whole until all increments are completed and a system-level test readiness review (TRR) is conducted. The resulting system could be deployed and operated providing new system requirements or changes in requirements for the next evolutionary iteration of the system.

The requirements and architecture framework, within each evolutionary iteration, is taken to be stable to facilitate the partitioning of contiguous requirement sets for increments based on some criteria as follows (Fairley, as cited in Pyster & Olwell, 2013): (1) priority of features; (2) safety-critical first; (3) user-interface first; and (4) kernel first followed by utilities.

The benefits of the Evolutionary and Iterative Vee are: (1) each increment has a tight build-verify-validate-demonstrate cycle which is quick to identify rework and fix defects (Pyster & Olwell, 2013); (2) flexibility to incorporate in-scope changes to requirements in subsequent iterative builds (Pyster & Olwell, 2013); and (3) long-term flexibility to incorporate scope-changing requirements in subsequent evolutions.

e. Spiral

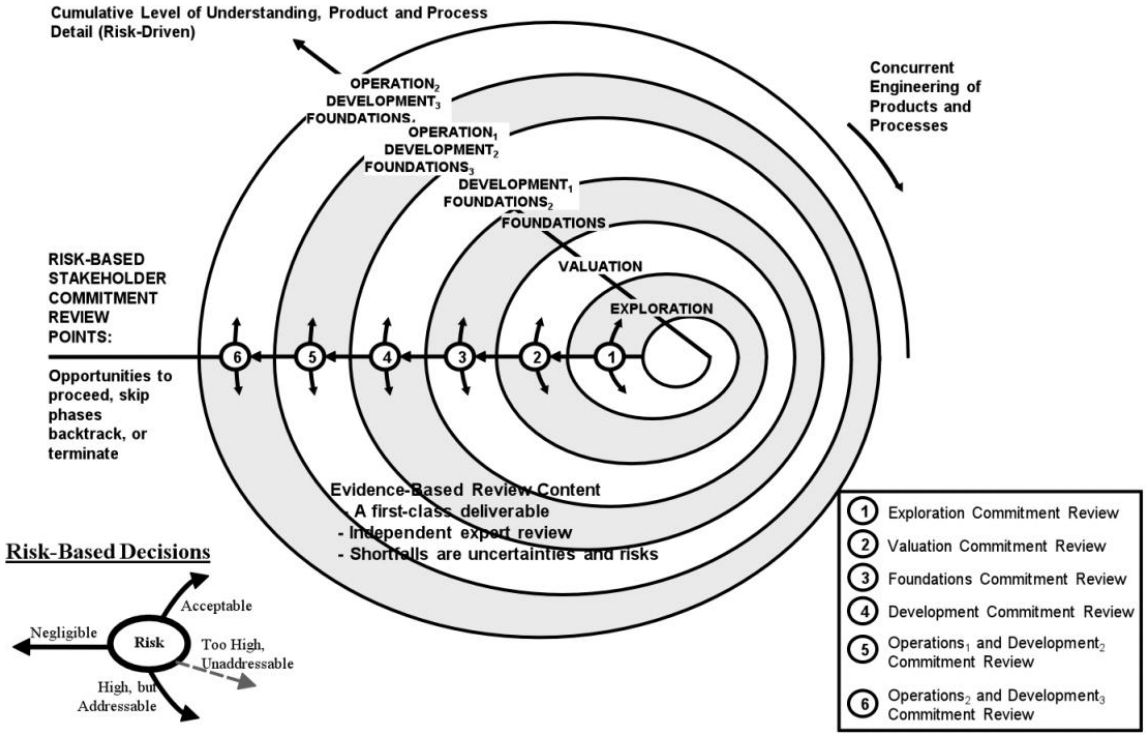


Figure 13. The Incremental Commitment Spiral Model (From Koolmanojwong, 2010).

The Spiral model is considered a primarily evolutionary and concurrent process model (Pyster & Olwell, 2013; Langford 2013b). Boehm was credited with the introduction of the Spiral model to SE in 1998, which has since been updated to the Incremental Commitment Spiral Model (ICSM) with the six risk-based decision reviews shown in Figure 13 (Koolmanojwong, 2010). For the purpose of this thesis, the Spiral process model refers to this updated ICSM model. The Spiral Model in Boehm's (2000) words with emphasis intact:

> The spiral development model is a ***risk-***driven ***process model generator***. It is used to guide multi-stakeholder concurrent engineering of software intensive systems. It has two main distinguishing features. One is a ***cyclic*** approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of ***anchor point milestones*** for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions

The main benefit of the Spiral model, apart from its ability to evolve a system based on changing stakeholder needs, is being able to lower cost by eliminating infeasible solutions earlier and avoiding rework (Boehm, 2000; Langford, 2013b).

Risks are “events that can cause the system to fail to meet its goals” (Boehm, 2000), and are ranked in terms of their combined impact and likelihood. Risks are addressed by prototyping, modeling, and trade-studies. During risk analysis, key characteristics of the system are determined and referred to as process drivers (Langford, 2013b).

The Spiral model is a process model generator, which allows capability delivery organizations to embark on either “an incremental, waterfall, evolutionary prototyping, or other subsets of process elements in the spiral model” (Boehm, 2000) based on the risk patterns identified. The process model generated would inform the organizations what should be done next and for how long (Boehm, 2000).

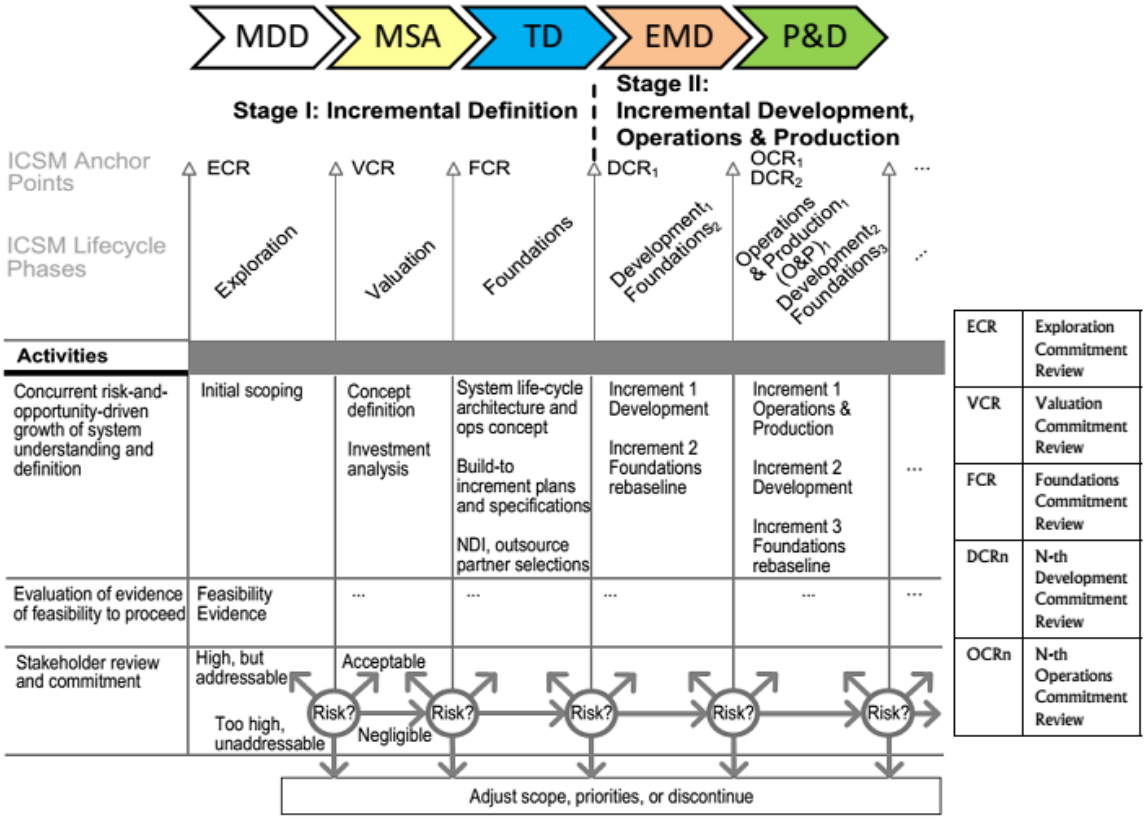


Figure 14. ICSM phase view with DAMS phases (After Koolmanjwong, 2010).

The Spiral model is the first concurrent process model encountered in this thesis, and it differs from the sequential process model in that life cycle issues are considered together instead of considering them sequentially, and so any stakeholder or engineer interested in providing a requirement or design input can do so at any point in the process (Langford 2013b) as shown in Figure 14.

A successfully executed Spiral process model would invariantly display the following six characteristics (Boehm, 2000): (1) concurrent rather than sequential determination of artifacts; (2) consideration of spiral elements[6] in each spiral cycle; (3) level of effort for activities commensurate with risk; (4) level of

6 Spiral elements are (1) critical-stakeholder objectives and constraints; (2) product and process alternatives; (3) risk identification and resolution; (4) stakeholder review; and (5) commitment to proceed (Boehm, 2000).

detail for artifacts commensurate with risk; (5) managing stakeholders' continued commitment through three anchor point milestones[7]; and (6) emphasis on life cycle rather than initial development.

f. Agile

The Agile method *is not a process in itself, but a method to be employed within a defined SE process model* (Pyster & Olwell, 2013). The Agile method could be used in an evolutionary process model (Pyster & Olwell, 2013). The process model employing the Agile method could also be sequential, opportunistic or concurrent depending on the system partitioning needs and technical competency of the capability delivery organizations with the agile process (Fruhling & Tarrel, 2008).

The *INCOSE SE Handbook* (2010) states that project execution methods can be described on a continuum from "adaptive" to "predictive" and agile methods are on the "adaptive" end. Despite being adaptive in nature, agility is neither "unplanned" nor "undisciplined" (INCOSE, 2010). It is guided by four key values in the Agile Manifesto (Bent et al., 2001):

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

An Agile process flow usually follows a Scrum method, as depicted in Figure 15, where there are two main threads of SE activities at a given point in time.

[7] Three anchor point milestones: (1) Life Cycle Objectives (LCO), (2) Life Cycle Architecture (LCA), and (3) Initial Operational Capability (IOC).

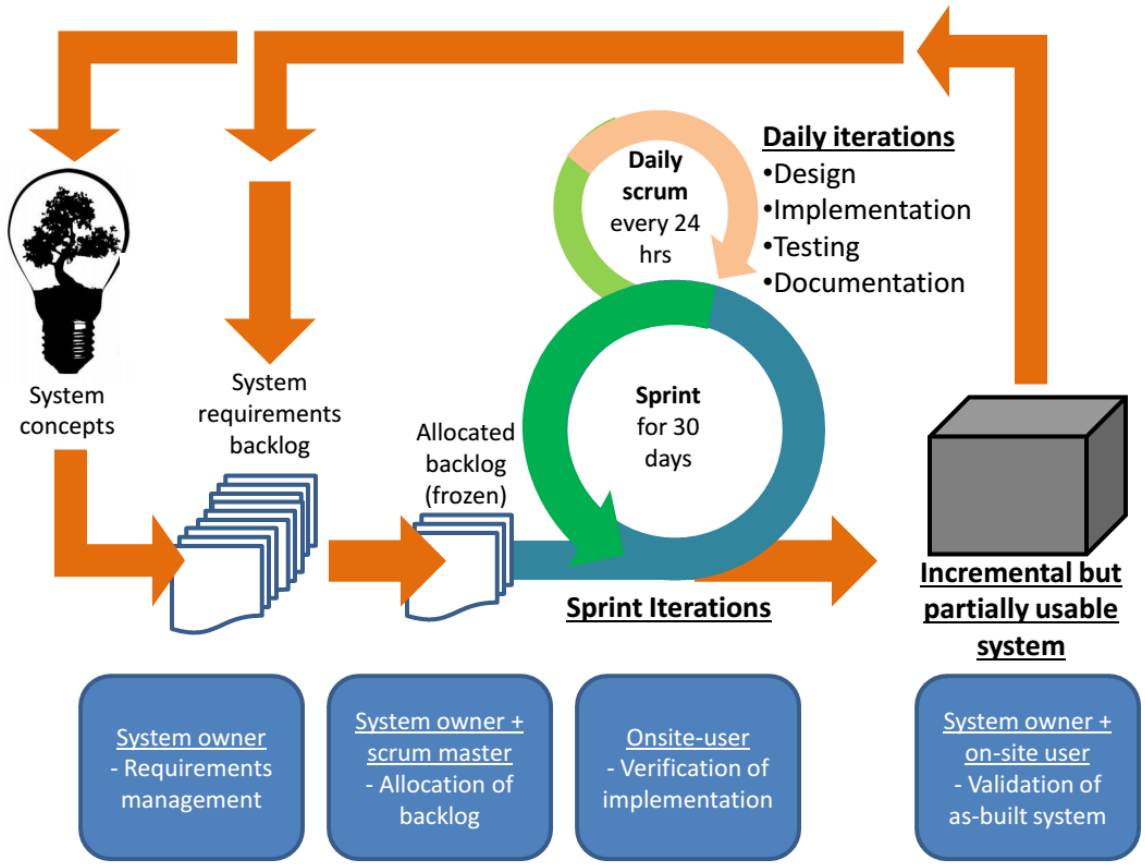


Figure 15. Agile model (After Boehm & Turner, as cited in Pyster & Olwell, 2013 and Fruhling & Tarrel, 2008).

The first thread is the Scrum thread that starts as a list of system backlog requirements is allocated by the system owner and scrum master to the team for a sprint (usually thirty days) (Fruhling & Tarrel, 2008). These allocated requirements are frozen for the duration of the sprint (Fruling & Tarrel, 2008). The scrum master expands the requirements into discrete tasks to be further allocated to the team which would then implement them with a daily rhythm that involves a scrum meeting (Pyster & Olwell, 2013). The meeting called by the scrum master allows every team member to provide a short update on what has been done, what problems they have, and what they would do before the next meeting (Fruhling & Tarrel, 2008; Pyster & Olwell, 2013). The Scrum thread ends with the sprint, and usually results in the delivery of a system with incremental functionality.

The second thread is the System backlog management thread (depicted by the orange arrows in Figure 15). The system owner controls the backlog through the scrum master, and ensures the central and continual re-prioritization of the requirements (Fruling & Tarrel, 2008). The requirements could be changed or added due to the requirements being deemed invalid through hands-on with the incremental version of the system or as discovered by the team during scrum sprints (Fruling & Tarrel, 2008).

These two threads could either happen sequentially or concurrently, depending on the team's grasp of the requirements. If the requirements are not well-defined, the team should perform these threads sequentially to leverage on the down-time between sprints to seek clarity on the requirements (Fruhling & Tarrel, 2008). Advanced teams, could perform these threads concurrently, and may even have multiple Scrum threads developing multiple incremental versions of the system concurrently (Fruhling & Tarrel, 2008).

With respect to the DAMS life cycle phases, the Agile method is suitable to be used within SE processes during the Technology Development phase and the Engineering & Manufacturing Development phase as the frequent and rapid builds would be too expensive during and after the Production & Deployment phase.

g. Wave

The Wave process model is a meta-process model view of the Trapeze model developed specifically for use with the delivery of acknowledged SoS capabilities. Dahmann et al. (2011) pointed out that many other SE process models were predicated on the capability delivery organization's ability to "define boundaries and requirements clearly and to control the development environment so that requirements can be optimally allocated to components," a premise that is no longer valid in an SoS environment.

The Trapeze model is an SE model for SoS and identified seven core SoS SE elements (ODUSD [A&T] SSE, 2008): (1) translating capability objectives, (2) understanding systems and relationships, (3) assessing performance to capability objectives, (4) developing and evolving an SoS architecture, (5) monitoring and assessing changes, (6) addressing requirements and solution options, and (7) orchestrating upgrades to SoS.

The Wave model shown in Figure 16 is a process-space visualization, with the model showing the time-sequenced steps overlaid on the needed SoS SE elements from the unwrapped Trapeze model on the left (Dahmann et al., 2011). "The arrows between the wave model elements depict the normal process flow, and the embedded circles in the arrows indicate that there may be and usually is back-and-forth iteration between these elements" (Dahmann et al., 2011).

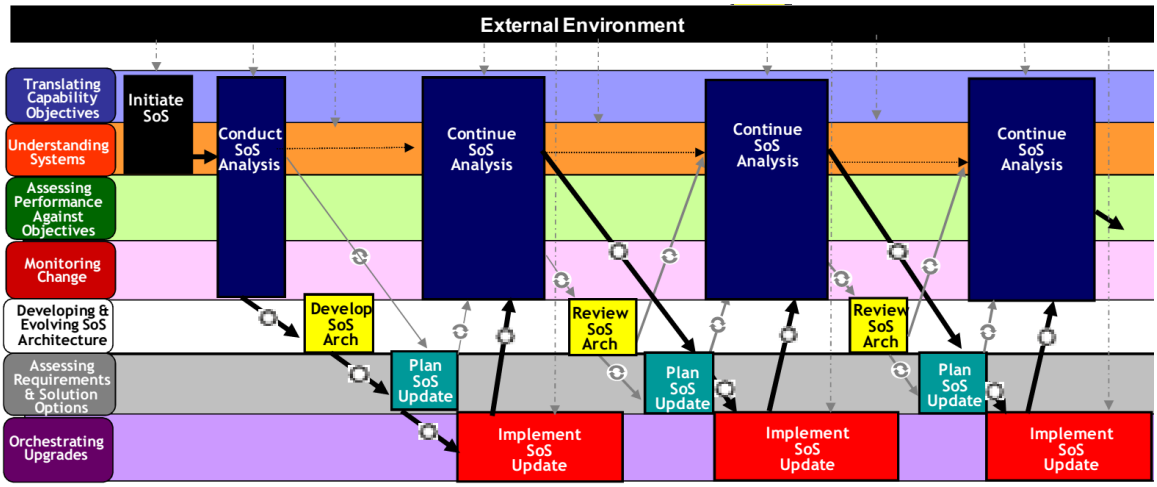


Figure 16. The Wave Model with the Unwrapped Trapeze Model (From Dahmann, Rebovich, Lowry, Lane, & Baldwin, 2011).

Dahmann et al. (2011) suggest that the Wave model is suitable for SoS SE as it has several characteristics that reflect the attributes of SoS: (1) multiple overlapping iterations of evolution; (2) ongoing analysis; (3) continuous input from external environment; (4) architecture evolution; and (5) forward movement with feedback.

Table 8 shows the additional information artifacts required by the Wave meta-process model as the capability delivery organizations go through the six steps of the model: (1) initiate SoS; (2) conduct SoS analysis; (3) develop and evolve SoS architecture; (4) plan SoS update; (5) implement SoS update; and (6) continue SoS analysis.

Table 8. Information artifacts for the six steps of the Wave Model (From Dahmann et al., 2011).

Information Applied at Each Step		Initiate SoS	Conduct SoS Analysis	Develop SoS Arch	Plan SoS Update	Implement SoS Update	Continue SoS Analysis
SoS Capability-Related Info	SoS Capability Objectives	Established	Used	Used Indirectly	Used	Used	Updated
	SoS Level CONOPS	Created	Used	Used Indirectly	Used	Used	Updated
	Requirements Space		Established	Updated	ID'd and Tagged	Updates	Updated
System Info	Systems Information	Captured	Used	Used	Used	Used	Updated
SoS Technical Information	Performance Measures		Defined	Used	Used	Used	Updated
	Performance Data		Captured	Used	Used	Updated	Updated
	SoS Technical Baselines		Created	Used	Created	Created	Updated Used
	SoS Architecture			Documented Updated	Used	Used	Used Indirectly
SoS Planning and Management Information	SE Planning Elements		Established	Updated	Updated	Updated	Updated
	Master Plan		Established	Updated	Updated	Updated	Updated
	Agreements		Established	Updated	Updated	Updated	Updated
	Technical Plan(s)			Evaluated	Established	Updated	Used
	IMS			Used	Established	Updated	Used
	Risks and Mitigations	Captured	Updated	Updated	Updated	Updated	Updated

E. CAPABILITY DELIVERY ONTOLOGY

The U.S. DoDAF is an "overarching, comprehensive, and conceptual model" that supports six core processes pertaining to capability delivery: (1) JCIDS process that ensures warfighters receive the capabilities required to execute their assigned missions successfully; (2) the DAMS management framework that translates mission needs into operational capabilities through a series of milestones; (3) the use of Systems Engineering that is required by DoD acquisition policies to establish a holistic life cycle perspective of the SoI; (4) program management processes; (5) Portfolio Management through architectural description; and (6) business & mission operations (DoD Chief Information Officer, 2012).

Vitech's *CORE 8 Architecture Definition Guide* describes a working ontological model based on DoDAF Version 2.0 (Vitech, 2011). This section reviews a subset of the model selected for the purpose of describing the key concepts relating to JCIDS's capability needs, DAMS life cycle phases, Systems Engineering and Program Management through the use of SEP models strategies to guide capability delivery.

In order to describe how a capability is taken through its life cycle by the capability delivery organizations, the ontology captures the three complementary domains relating to the: (1) capability's operational architecture; (2) system architecture; and (3) program management as shown in Figure 17. The operational architecture documents the envisaged capability's operational concepts through its activities, tasks, and military performers. The system architecture records the corresponding requirements on the functions of a physical component.[8] The program management domain tracks the program activities needed as per the capability's life cycle phase and the program

[8] A component in the CORE ontology is recursive composite entities that represent an SoS or system type (Vitech, 2011).

elements[9] needed to accomplish it. The three domains are intricately woven together, with the operational architecture defining solution-agnostic functions to support their capability needs, the systems architecture defining the system components that would perform these functions, and the program management executing the program elements needed to supply the system components.

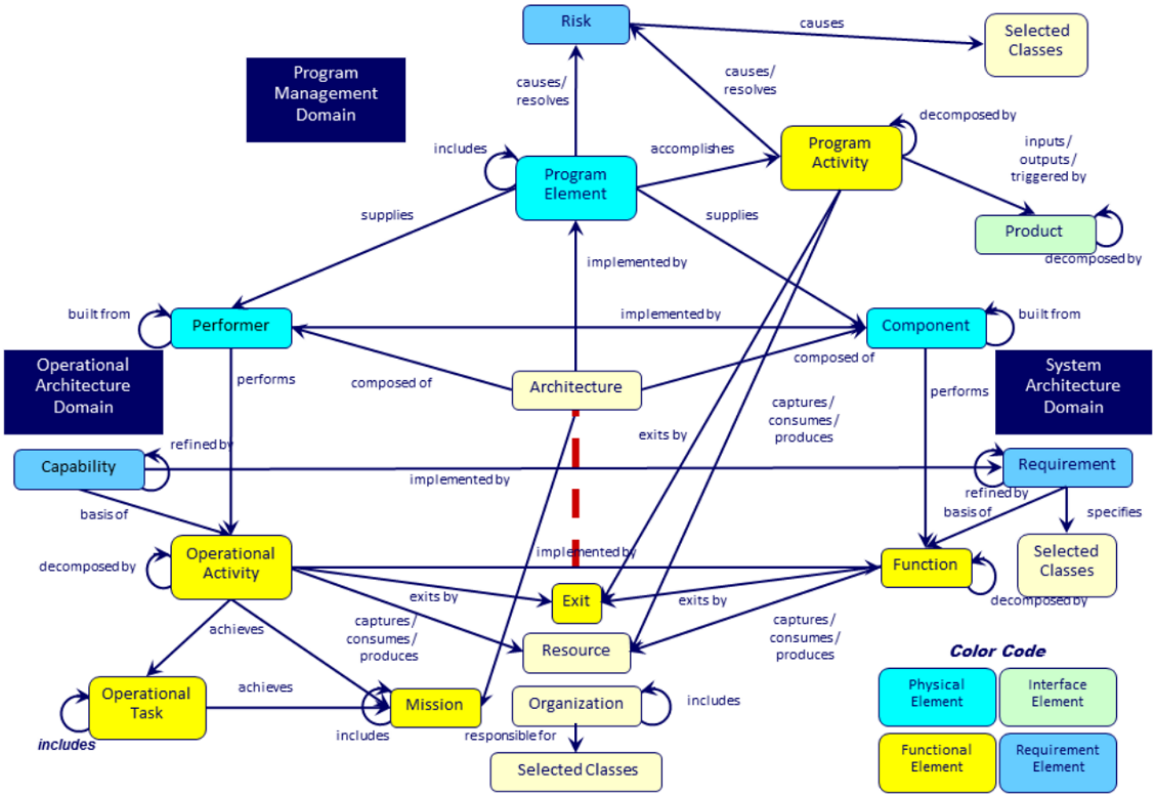


Figure 17. Operational, system & program domains in CORE's Schema (From Vitech Corporation, 2011, used with permission).

The program management domain could be further elaborated as shown in Figure 18, whereby a capability delivery organization could be responsible for the various ontological entities as shown. Of particular interest to this thesis is the organization's assigned responsibility for a program activity (such as a DAMS life cycle phase or step in a process model), and the

[9] The program element in the CORE ontology is a recursive composite entity that represents either a program, project, work package or task (Vitech, 2011).

organization's assigned program elements (work packages) to implement the architecture or the physical components (parts and whole of an SoI) of a capability.

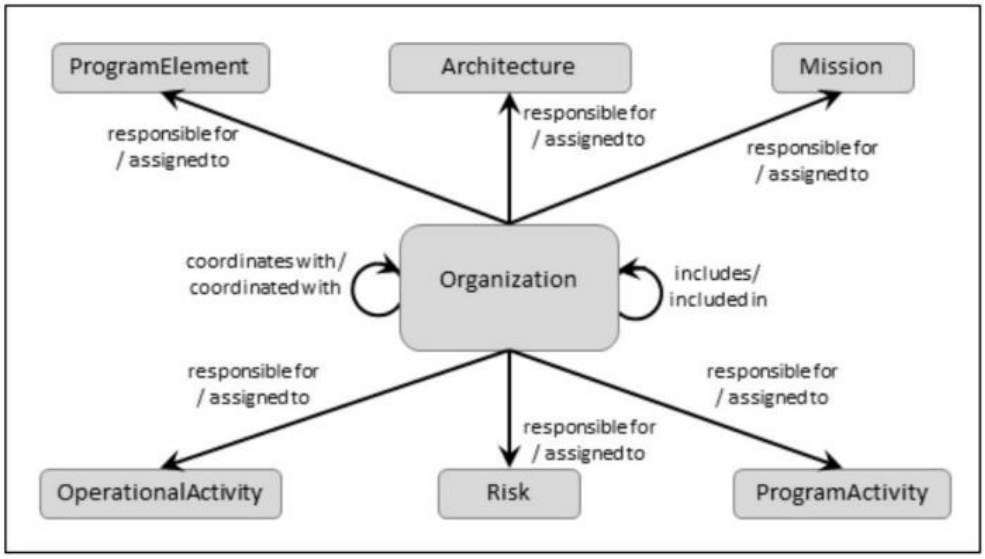


Figure 18. Model of capability delivery organization in CORE's Schema (From Vitech Corporation, 2011, used with permission).

The CORE 8 ontology represents a working ontology that has been rigorously improved and used for model-based systems engineering. The CORE 8 ontology is chosen as the base over which an extended CDS ontology would be proposed because it is familiar to both defense academics and practitioners. The CDS ontology with emergence developed in this thesis would be more readily adapted or extended in part or whole to other DoDAF compatible research.

III. RESEARCH APPROACH

This chapter discusses the scope of the research pertaining to the capability delivery system, and the overall approach to achieve the research objectives provided in Chapter I.

A. OVERVIEW

- Section B scopes the CDS and identifies inputs and noise factors to which the CDS is subject.
- Section C presents an extended capability delivery ontology with the central theme of emergence.
- Section D describes how the ontology would be used by the CDS through the DAMS life cycle phases.
- Section E specifies how a subset of the SE process model strategies would influence the generic flow of DAMS events and work packages.
- Section F elaborates on the potential input, control and output variables for a CDS Simulator (CDSS).
- Section G explains the impetus behind developing a CDSS based on the proposed ontology with emergence.

B. SCOPE OF THE CAPABILITY DELIVERY SYSTEM

The earlier sections covered the capability-based approach towards force modernization supported by the JCIDS, the differences between systems and SoS, how desired capabilities are implemented based on desired emergent traits of the SoI, the DAMS working model of a system's life cycle, and the use of a variety of SE processes to help manage the complexity of delivering modern systems.

Figure 19 shows a high level map of how the various concepts come together to form a meta-model of capability delivery. A capability need is conceived by its sponsors and put through the JCIDS. JCIDS would assist the JROC and the corresponding MDA to decide if this need is valid and warrants a materiel solution. If so, the MDA would then decide the appropriate entry

milestone depending on the technological and operational maturity of the materiel solution. The capability enters the DAMS at the milestone designated by the MDA after a positive MDD and is taken through its life cycle, gradually maturing the SoI that would implement the capability until its deployment. The SoI would be realized by many capability delivery organizations that change along its life cycle as depicted at the bottom of the figure. The organizations interact with each other based on the chosen SE process to satisfy DAMS requirements. The black block that separates these organizations from the maturing SoI represents the organization's fog of emergence that could influence the perception of the system's achieved performance and their engineering decisions.

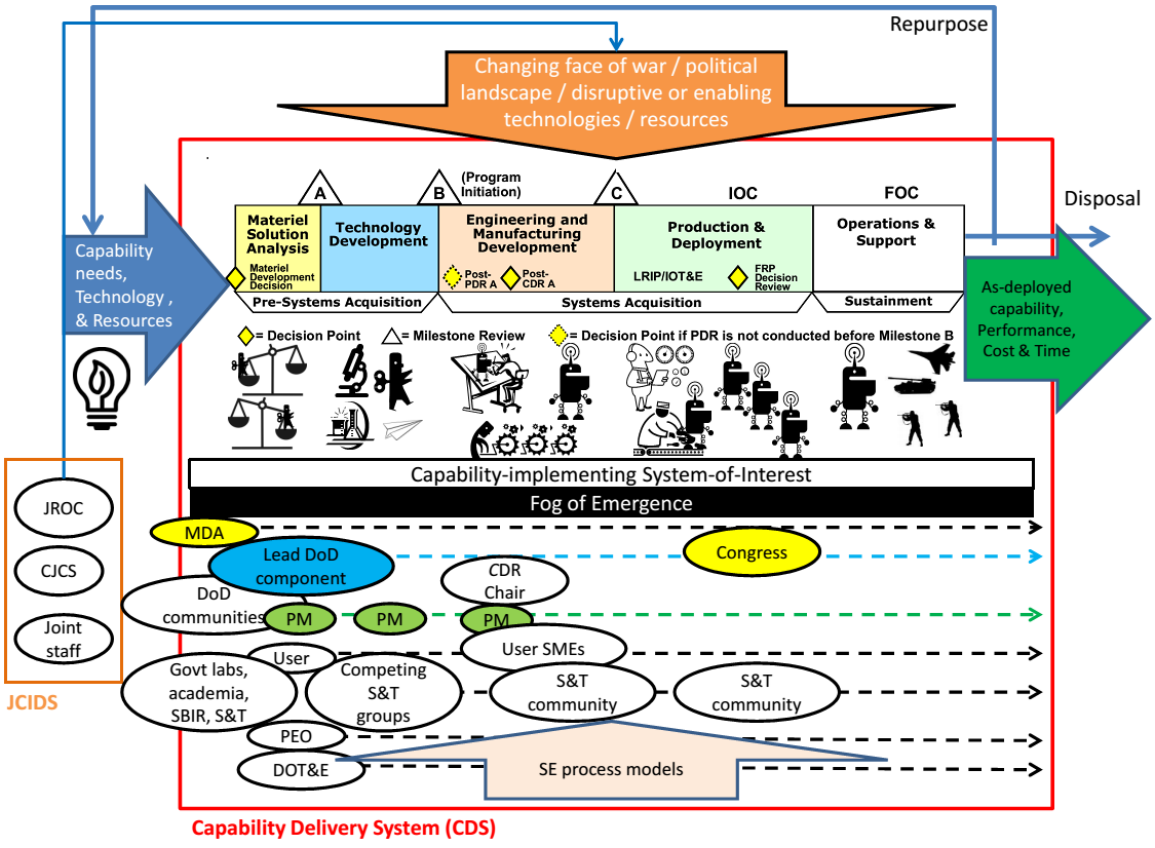


Figure 19. Conceptual scope for a Capability Delivery SoS (After OUSD AT&L, 2008).

Figure 19 shows a high level map of how the various concepts come together to form a meta-model of capability delivery. A capability need is conceived by its sponsors and put through the JCIDS. JCIDS would assist the JROC and the corresponding MDA to decide if this need is valid and warrants a materiel solution. If so, the MDA would then decide the appropriate entry milestone depending on the technological and operational maturity of the materiel solution. The capability enters the DAMS at the milestone designated by the MDA after a positive MDD and is taken through its life cycle, gradually maturing the SoI that would implement the capability until its deployment. The SoI would be realized by many capability delivery organizations that change along its life cycle as depicted at the bottom of the figure. The organizations interact with each other based on the chosen SE process to satisfy DAMS requirements. The black block that separates these organizations from the maturing SoI represents the organization's fog of emergence that could influence the perception of the system's achieved performance and their engineering decisions.

The red box shows the scope of the Capability Delivery System (CDS). The JCIDS is considered to be out of scope, but it provides important: (1) initial inputs; and (2) noise factors to the CDS. The initial inputs are the capability needs, available technology, and resources for the CDS. A necessary and logical outcome of a capability-based approach facilitated by the JCIDS means that the changing face of war, political landscape, and technologies could trigger a modification, invalidation, or insertion of requirements even while the SoI was undergoing development. These enter the CDS as noise factors that are beyond the control of the CDS, but must be dealt with in order to ensure that the capability as-deployed matches the relevant capability requirements at the time. These input and noise capability factors are assumed to be accompanied by a positive MDD. Any capability need that is unable to obtain the MDD would not be able to enter the CDS and hence will not be a factor.

The systems engineering processes selected by the organizations to help realize the SoI, would help them decide what to do and for how long in accordance to DAMS deliverables, as well as analyze, detect, and know about the SoI's full suite of emergent traits.

C. CAPABILITY DELIVERY ONTOLOGY WITH EMERGENCE

The CORE 8 ontology reviewed in Chapter II was extended to reflect the axiomatic concepts regarding the fog of emergence; specifically the difficulty experienced by capability delivery organizations to know the actual extent of an SoI's many emergent traits due to both the indeterminate nature of some emergent traits, multiplicity of contexts (especially divergence between context designed for and the actual contexts in which the SOI would be deployed), and system darkness.

The fog of emergence as shown in Figure 20 is a representation of the organization's knowledge of emergence as a state transition diagram. Simply put this diagram shows that a capability delivery organization's knowledge of any emergent trait is subjective and runs a gamut of no knowledge to full knowledge against an external objective manifestation of that same trait, which is intrinsically exhibited by the functions performed by a SoI component.

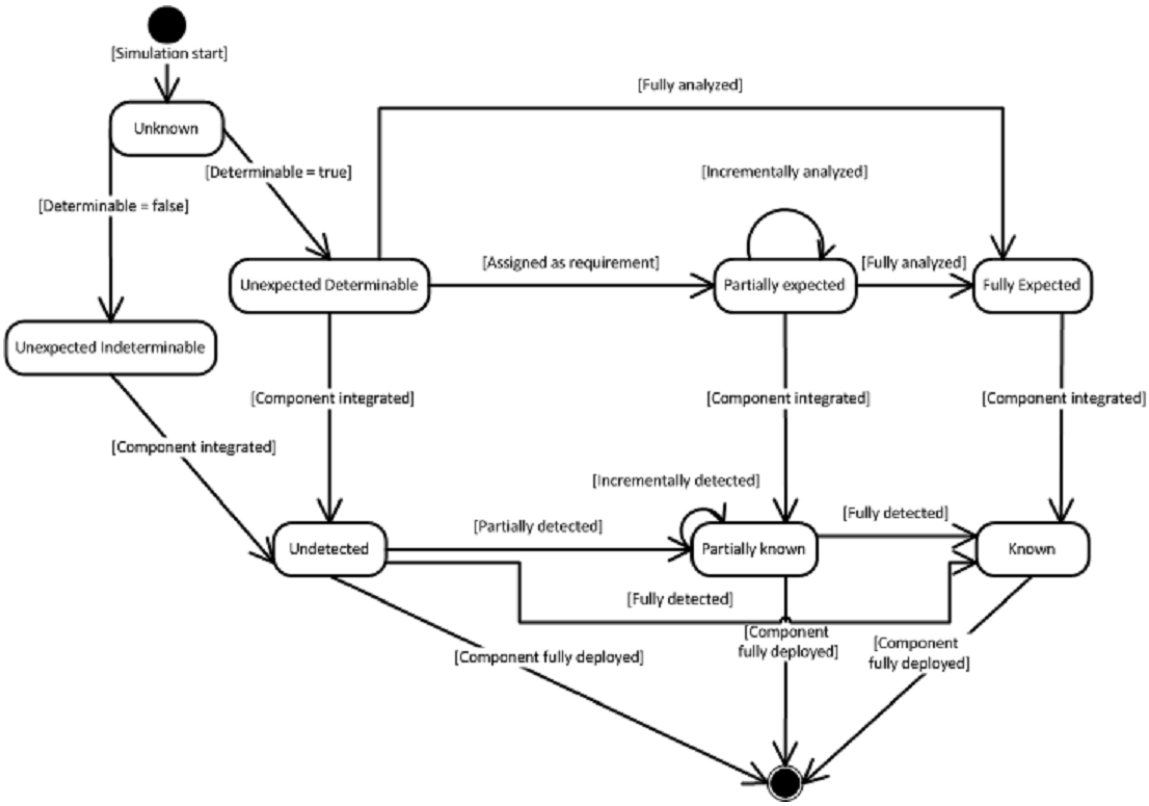


Figure 20. Fog of Emergence.

Figure 20 shows that an initially “Unknown” emergent trait exhibited by a function performed by a given component (part or whole of a SoI) could be classed as either an “Unexpected Indeterminable” or “Unexpected Determinable” emergent trait. Recall that due to system darkness and multiplicity of contexts for the as-deployed SoI, where on one hand, an emergent trait that is “Indeterminable” is one that could only be detected together with the operation of the SoI in its multiple as-deployed operational contexts. “Determinable” emergent traits, on the other hand, arise due to the interaction of SoI parts in the intended Mission context.

If an emergent trait is assigned as a desirable functional requirement or analyzed in its as-intended context, it becomes “Partially Expected.” A “Partially Expected” emergent trait could occasionally become “Fully Expected” through a

successful and complete analysis of the mechanisms through which the emergent trait arises.

After the component is integrated, the fog of emergence for that emergent trait automatically transits the various states of expectation to the corresponding states of knowledge as shown in Figure 20. The inherent simplifying assumption here is that we assume that when an emergent trait is expected, the organizations would be in a position to employ the appropriate methods, tools, and measures to detect and hence know about it after the component is integrated.

Similarly, the transition from a lower state of knowledge of an emergent trait to a higher state knowledge of emergent trait occurs when the organization detects more of the exhibited emergent trait through the testing or operation of the component.

Finally, Figure 20 shows that the fog of emergence may never be lifted fully when the SoI is deployed, especially if the various organizations are no longer actively seeking to discover and measure potentially unknown emergent traits until these traits result in a delayed systemic failure of the system.

The transitions from lower states of knowledge to higher states of knowledge are influenced by the competency level of the organization to analyze and detect this trait using the appropriate methods, tools and measures. Specifically, for each emergent trait, there is an engineering sub-domain and a corresponding level of competency in that domain to be able to analyze and detect it. An organization that has a low level of competency would logically have a lower chance of analyzing or detecting the emergent trait even if the trait was manifested and experienced during operation.

The extended CDS ontology with emergence that is consistent with the narrative of how the fog of emergence could be lifted by Organizations during capability delivery is shown in Figure 21. The new entities of Fog of Emergence, Emergent Trait, Mission Context, and their associated links are highlighted in red.

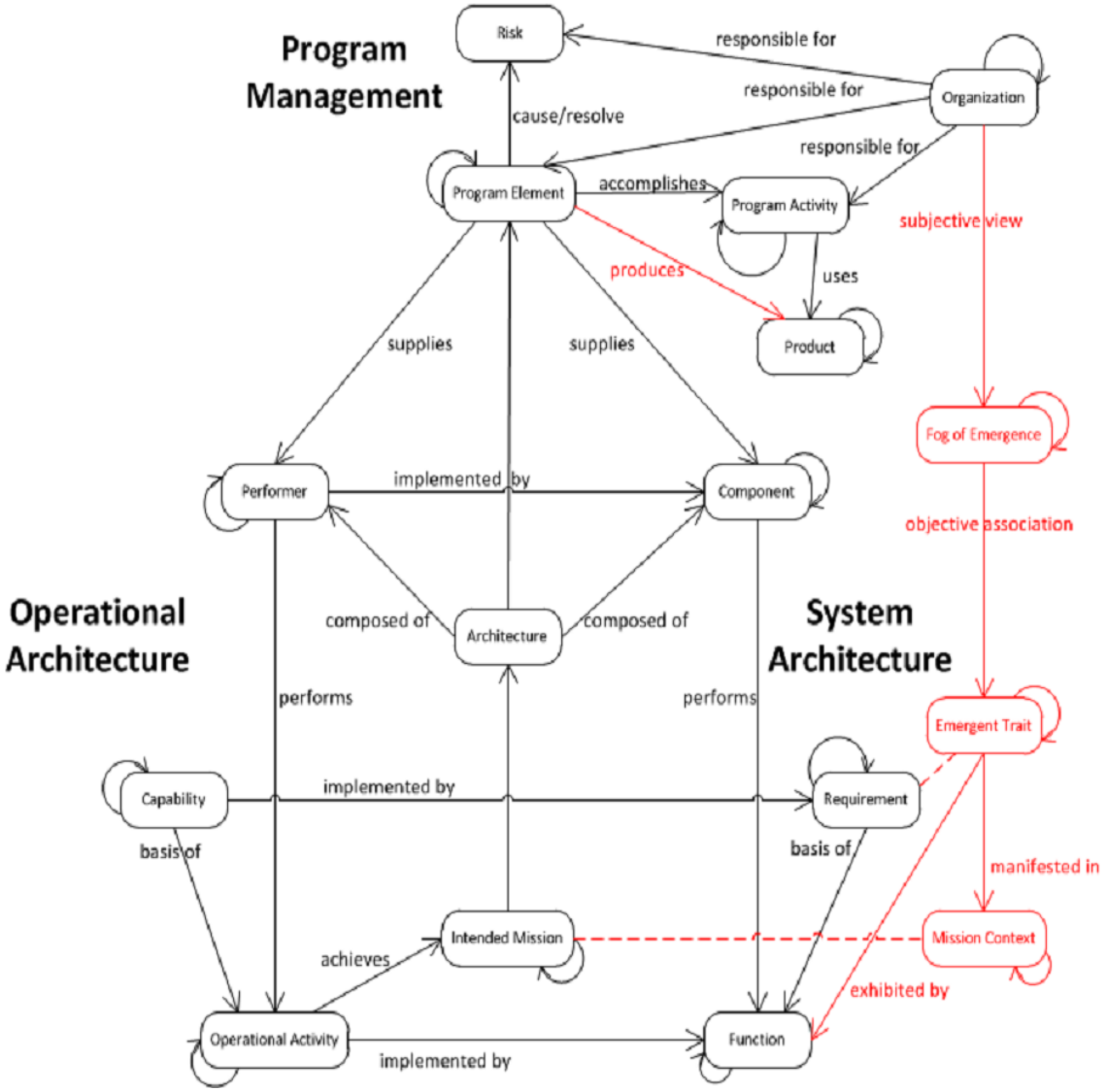


Figure 21. Extended CDS ontology with emergence (After Vitech Corporation, 2011, used with permission).

An Organization's perception of the SoI's performance (achievement of Requirements for Functions performed by Components) is always obscured by the Fog of Emergence. The dotted lines between: (1) Emergent Trait and Requirement; and (2) Mission Context and Intended Mission, show a *proper* subset relationship. The set of Requirements desired of an SoI's Function is a proper subset of the full suite of Emergent Traits exhibited by the same Function. The Intended Mission is a proper subset of the multiplicity of Mission Context that

the SoI would be tasked to undertake in its lifetime regardless of design intentions. An Organization's natural propensity to focus on the Requirements and Intended Missions give rise to the Fog of Emergence, where the Organization only sees a subjective view of the objective reality.

D. USE OF CDS ONTOLOGY BY THE CDS

1. Narrated Walk-Through

The following narrative walks through how the CDS ontology with emergence would be used to capture the evolving products and SoI components of the capability and its delivery by the organizations responsible, with the entities of the ontology capitalized **bolded** for emphasis.

During the early phases of capability delivery, the JCIDS match capability needs to capability providers. The input for the CDS comes in the form of an **Operational Architecture**. The **Operational Architecture** captures the **Capability** needed which forms the basis for an **Operational Activity** to be performed by a military **Performer** to achieve an **Intended Mission**.

The CDS starts with the MSA **Program Activity**, where the **Program Elements** comprise conceptualization of operations and developing a value system for the stakeholders to perform an Analysis of Alternatives (AoA). These are represented by the refinement of **Operational Activities** and **Intended Mission** contexts, which helps identify the implementing **Functions**. The stakeholders would then define a measure for the satisfactory performance of the **Function** as a **Requirement**. The Analysis of Alternatives would use these **Requirements** to evaluate alternative **Components** in their performance of these **Functions** and culminates in a sub-**Program Activity** that reviews the readiness to complete MSA and proceed to TD **Program Activity**.

The TD **Program Activity** starts with the Milestone A sub-**Program Activity**. During the TD phase, the main **Program Elements** would involve refining the System Architecture by building a number of competing prototype **Components**. Using these prototypes during demonstrations, users could

identify insufficiencies in **Operational Activities** and **Intended Missions**, and competing vendors could assess how their prototype **Components** would perform when operated by the users against the desired **Requirements**. The **System Architecture** would continually be refined until a high-confidence system-level design of the baselined system is produced and captured in the Preliminary Design Review (PDR) report **Product**. The PDR report would be reviewed at the PDR sub-**Program Activity**, after which the Capability Development Document (CDD) **Product** would be prepared for the eventual transition to the EMD **Program Activity**.

The Milestone B sub-**Program Activity** starts the EMD **Program Activity**. There are two main threads of Program Elements, with the first being the Integrated System Design **Program Element** and the second being the System Capability & Manufacturing Process Demonstration **Program Element**. The Integrated System Design **Program Elements** straddle the design-review sub-**Program Activities**, where the capability delivery **Organization** would have a chance to analyze both their design **Products** and prototype **Components** for insufficiencies. With the design review sub-**Program Activities** completed, the System Capability & Manufacturing Process Demonstration **Program Element** begins. The **Program Elements** to build both the mission and support **Components** straddle the repeated DT&E sub-**Program Activities**. Through these activities, the capability delivery **Organization,** together with the user, would have a chance to assess the performance of the developed **Components,** and also identify new **Operational Activities** and new **Intended Missions** for the next iteration. When the **Systems Architecture** and developed **Components** were demonstrated to meet **Requirements** of **Functions** that implement the needed **Operational Activities** of a **Capability** for the **Intended Missions** during a DT&E **Program Activity**, a **Program Element** to produce the Capability Production Document (CPD) **Product** would begin.

The P&D **Program Activity** begins with the Milestone C sub-**Program Activity** to authorize entry into LRIP. The **Program Elements** to prepare the

minimum production baseline and production-representative **Components** would lead up into an IOT&E sub-**Program Activity** where the mission and support ***Components*** would be evaluated in their **Intended Mission** contexts. A satisfactory IOT&E sub-**Program Activity** would lead to preparation of a "Beyond LRIP[10]" **Product** and FRP Decision Review sub-**Program Activity**. **Program Elements** to execute FRP and deployment of **Components** would follow and lead up to successive FOT&E sub-**Program Activities**. When the full SoI **Components** have been evaluated satisfactorily against the **Requirements** of **Functions** that implement the needed **Operational Activities** of a **Capability** for the **Intended Missions**, **Program Elements** to perform military equipment valuation would begin to transit to the O&S **Program Activity**.

The O&S **Program Activity** is marked with continual SoI **Component** operations and support **Program Elements** and broken up by occasion repeated review sub-**Program Activities**. This is where the previously unintended **Mission Contexts** would start emerging and where previously unanticipated **Emergent Traits** would manifest. However, even if the **Emergent Traits** manifest, they might not be detected as the capability delivery **Organizations** only know about these traits through a competency dependent **Fog of Emergence**.

Appendix A describes the same flow of **Program Activities**, sub-**Program Activities**, **Program Elements**, **Products**, **Systems Architecture** and SoI **Components** as described above in an N^2 chart format. This table shows the specific deliverables, and focus on different parts of the ontology through the various life cycle phases with processes such as **Program Activities** and **Program Elements** forming the diagonal spine of the N^2 chart, and objects such as **Products**, **Systems Architecture** and SoI **Components** as the output and input to the diagonal spine.

10 A "Beyond LRIP Report" provides the knowledge to support the MDA's decision to proceed beyond LRIP. The report captures knowledge that demonstrated control of manufacturing process and reliability.

Throughout the various **Program Activities** of the **Capability**'s life cycle, the higher the **Organization**'s competency level in the requisite engineering competencies to measure the **Emergent Trait**, the greater the chance to pick up on it. During **Program Activities** prior to any SoI **Component** being tested in its intended environment, the **Organization** has a chance to know of determinable **Emergent Traits** arising from analysis of design **Products**. After the SoI **Component** is built and tested in its operating environment, the **Organization** has a chance to know of previously indeterminable **Emergent Traits** arising from the use of the SoI **Component** both in its **Intended Mission** environment as well as as-deployed **Mission Contexts**.

2. Diagrammatic Relation Between the Ontology and DAMS Life cycle Phases

By aggregating the CDS ontological entities into the Operational Architecture, System Architecture, implemented SoI Component, we could map how these aggregated entities are affected by DAMS life cycle phases (which are main and sub-Program Activities) as shown in Figure 22.

The five DAMS life cycle phases and key sub-Program Activities are captured as rows. The column headers follow a generic problem solving process, where we: (1) analyze the problem to determine requirements; (2) design and architect a solution to those requirements; (3) develop and acquire a system to the design; and (4) integrate and test the system developed. The fifth column shows the main outcome for each corresponding row's flow through of the generic problem solving process.

These Program Activities result in the production and refinement of the corresponding CDS ontological entity in the final row; (1) Sub-Program Activities to perform Analysis and Requirements would produce or refine the Operational Architecture; (2) Design & Architecture sub-Program Activities would produce or refine the System Architecture; (3) Development & Acquisition sub-Program Activities would produce SoI Components built according to the System

Architecture; and (4) Integration & Test would result in SoI Components being validated against the Operational Architecture. The fifth column shows that the Fog of Emergence is the main outcome when these sub-Program Activities are performed with a focus on Intended Missions and Requirements instead of As-Deployed Mission Contexts and Emergent Traits.

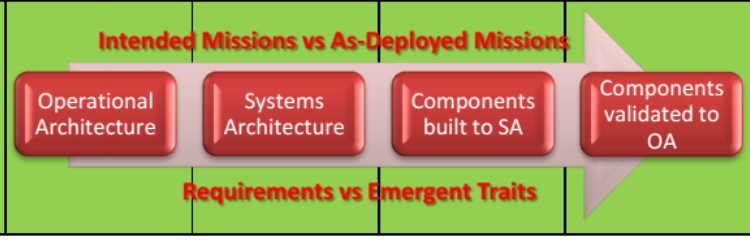
DAMS Lifecycle Phases	Analysis & Requirements	Design & Architecture	Development & Acquisition	Integration & Test	Outcome
MSA	Materiel Development Decision	Conceptual Design	Model-Based Systems Engineering	Analysis of Alternatives	Initial Capability Document based on AoA
TD	Milestone A	Prototype Preliminary Design	Competing prototypes components	Technological and Concept Demonstration	Capability Development Document
E&MD	Milestone B	Integrated Systems Design	Developmental system components	DT&E for System Capability & Manufacturing Process Demonstration	Capability Production Document
P&D	Milestone C	LRIP	Representative production article & manufacturing support	IOT&E	Beyond LRIP Report
	FRP Decision Review	FRP	Actual SoI components & production support	FOT&E	IOC
O&S	Lifecycle Sustainment	PEO Review	Iterative reviews	Operations & Maintenance	FOC
CDS Ontology 					**Fog of Emergence**

Figure 22. Relationship between the CDS Ontology and DAMS main and sub-Program Activities

Certain mapping of sub-Program Activities such as LRIP and FRP to "Design & Architecture" remains contrived, however, the overall mapping falls naturally in place and allows a reader to quickly grasp how each the DAMS life cycle phases would increasingly refine a Capability's Operational Architecture, System Architecture and SoI Components through a process of analysis-design-build-test.

E. INFLUENCE OF SE PROCESS MODEL STRATEGIES ON THE CDS

While seven SE process models have been reviewed, the SE process model strategies were encoded and used to influence the sequential or concurrent allocation of DAMS life cycle main and sub-Program Activities to deliver a capability using three common SE process models in this exploratory research: (1) the classic Waterfall, (2) the classic Vee, and (3) the Spiral.

1. Waterfall DAMS Strategy

As reviewed in Chapter II, the Waterfall SE process model is a pre-specified, single-pass, sequential and ordered process model. The allocation of DAMS life cycle main and sub-Program Activities (previously discussed in Figure 22) to the Waterfall process model would be a straightforward affair, where each program activity would happen in the sequence as captured with no additional SEP model-specific Program Activities.

A unique behavior of the Waterfall model would be that it is insulated from changing Capability needs, and the only type of rework it allows for would be for verification-type rework. Verification type rework would be rework due to erroneous work done for requirement engineering, system design, development and acquisition that cannot be verified against a precedent Program Activity artifact (for example erroneous requirements developed derived from capability needs captured in the MDD, or wrong implementation of a design based on an Integrated System Design document).

The DAMS life cycle Program Activity allocation strategy and SEP model specific behaviors are tabulated in Table 9.

Table 9. Waterfall DAMS Strategy

	Waterfall DAMS Strategy
Description	Pre-specified, single-pass, sequential and ordered
Sequential or Concurrent Allocation of DAMS activities	Sequential and one is to-one mapping of DAMS Program Activities for capability delivery
Behavior	Goes through corresponding DAMS phases in strict order
	Verification-type rework: Only does rework for erroneous work packages based on a fixed initial set of requirements derived from Capability needs.

2. Vee DAMS Strategy

The Vee model as reviewed in Chapter II is taken to be a pre-specified, sequential and ordered process model much like the Waterfall model, but with greater focus on the use of Systems Engineering, Verification and Validation (V&V), Decomposition along its process.

The DAMS life cycle Program Activities would be allocated in a one-to-one manner, with the flow between activities to be strictly sequential and in order. However, as there is additional focus on V&V by the Vee model, corresponding Program Activities would be added when going down the left of the Vee (Figure 23) and coming up the right of Vee (Figure 24).

When going down the left of the Vee, the Vee model follows a System Analysis and Design process to develop the Concepts of Operation and “Build To” specifications. When coming up the right of the Vee, the Vee model follows a System Verification and Integration process to verify correctness of implementation of sub-components to the specifications before approving the integration of sub-components into an aggregated component.

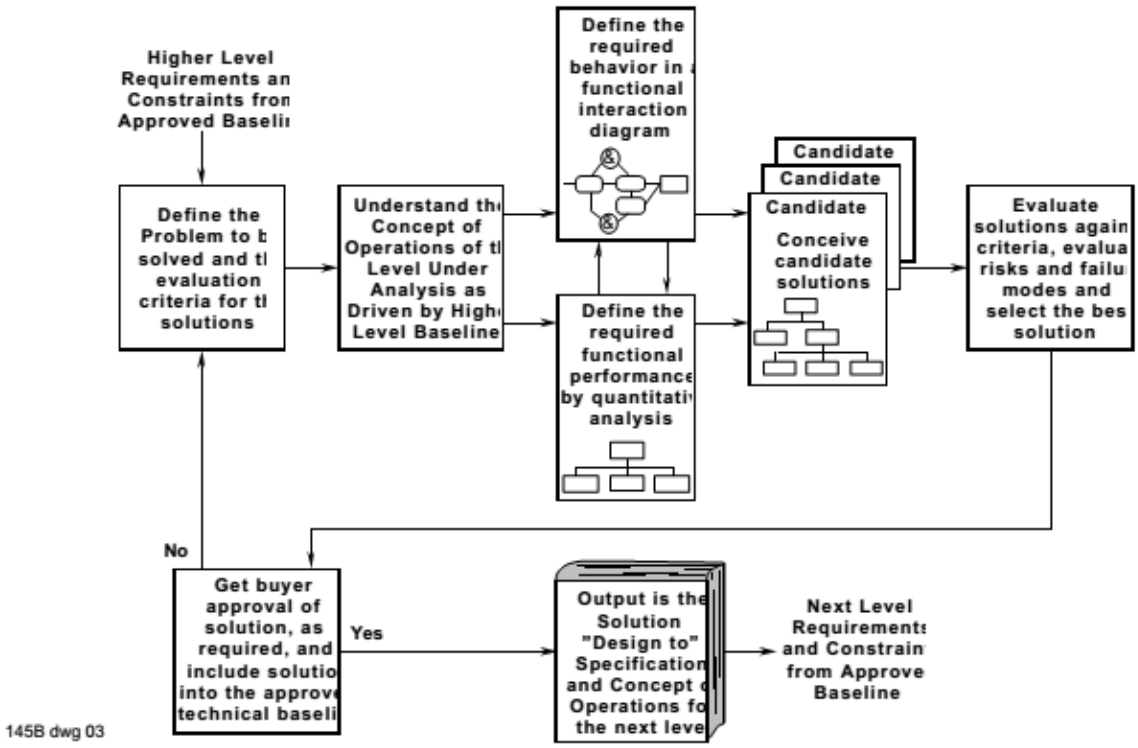


Exhibit 7—System Analysis and Design Process

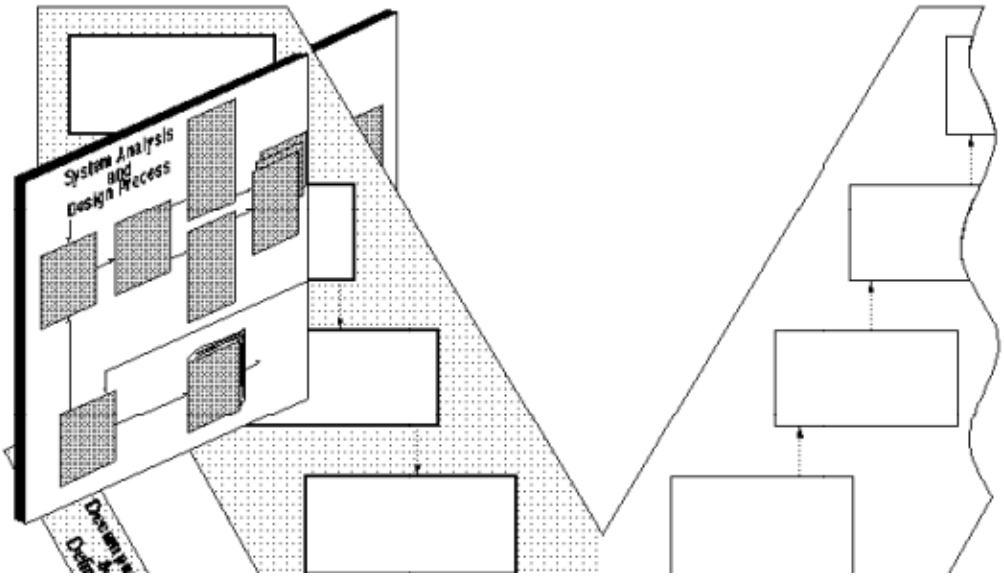


Figure 23. Application of the System Analysis and Design Process down the left of Vee[11] (From Forsberg & Mooz, 1995)

11 Figures 23 and 24 were cropped from Forsberg & Mooz 1995 paper. The truncation of text within text boxes was inherent from source.

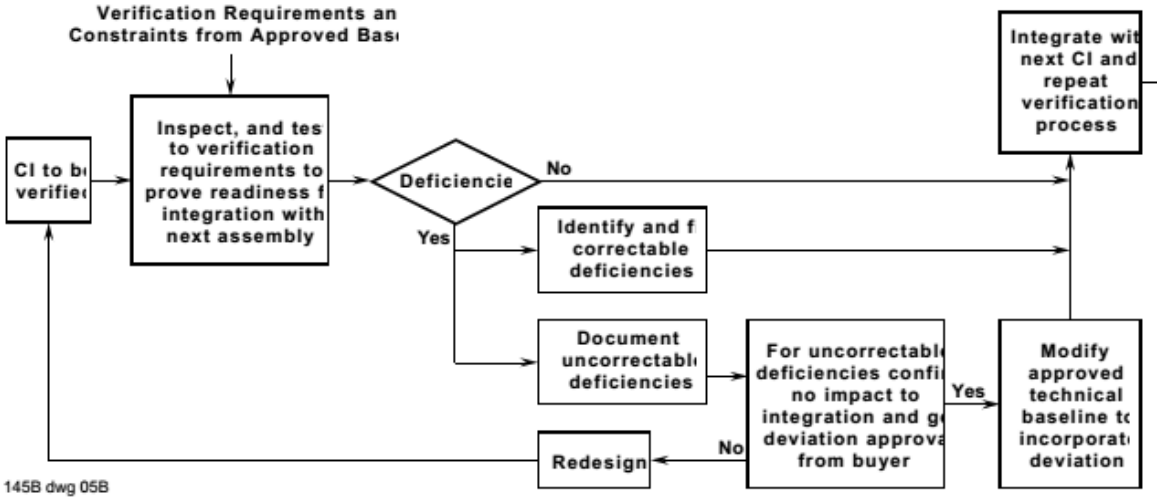


Exhibit 9—System Verification and Integration Process

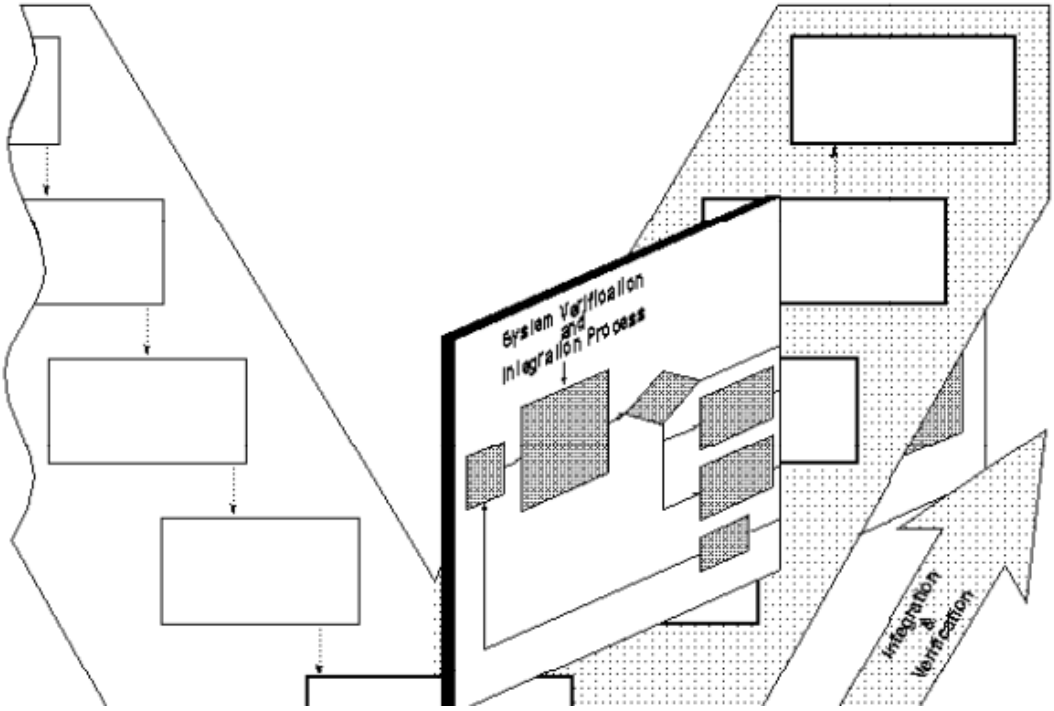


Figure 24. Application of the System Verification and Integration Process to the right of Vee (From Forsberg & Mooz, 1995)

Additional Vee model specific behavior is that it recognizes validation-type rework. Validation type rework is work done to rectify shortcomings in initial Capability Needs and intended Mission contexts.

The DAMS life cycle Program Activity allocation strategy and SEP model specific behaviors for the Vee model are tabulated in Table 10.

Table 10. Vee DAMS Strategy

	Vee DAMS Strategy
Description	Pre-specified, sequential, and ordered like waterfall with greater focus on SE, V&V, and decomposing/integration of system baselines
Sequential or Concurrent Allocation of DAMS activities	Sequential and one is to one mapping of DAMS Program Activities for capability delivery.
	Additional SEP model specific Program Activities would be added for going down the left of the Vee and coming up the right of the Vee.
Behavior	Goes through corresponding DAMS phases in strict order
	Does verification-type rework for erroneous work packages based on a fixed initial set of requirements.
	Allows validation-type rework if SoI does not work according to newly discovered mission contexts.

3. Spiral DAMS Strategy

The Spiral model as reviewed in Chapter II is an evolutionary and concurrent process model with greater focus on risk-based decision reviews.

The allocation of DAMS life cycle Program Activities for capability delivery can be templated based on Figure 25. The total CDS ontological entities could be partitioned into three contiguous sets to be delivered in three iterations. The Program Activities of the Spiral model has to arranged sequentially with some concurrent overlaps between iterations. The concurrent nature of the spiral model is not limited to overlaps between iterations, and as reviewed in Chapter II, manifests as overlaps between Program Activities within an iteration as shown in Figure 14.

A unique behavior of the Spiral model is its process generation behavior, which means that when the context suits the use of another process model, the

Organization could choose to deliver a particular increment using another process model.

The DAMS life cycle Program Activity allocation strategy and SEP model specific behaviors for the Spiral model are tabulated in Table 11.

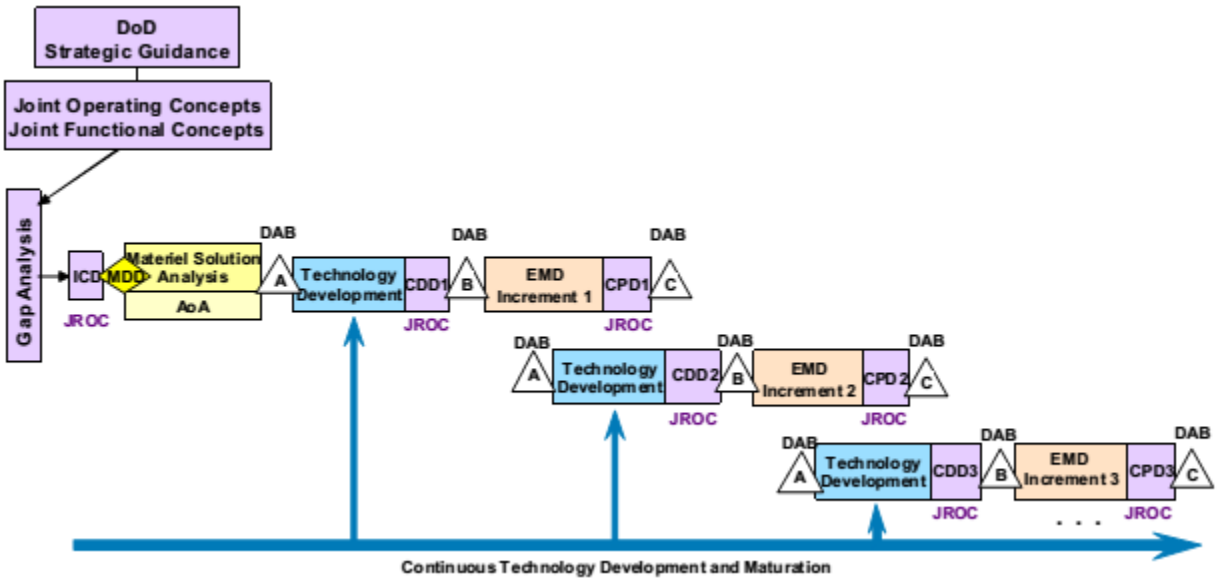


Figure 25. Concurrent development between iterations of spirals (From OUSD AT&L, 2008)

Table 11. Spiral DAMS Strategy

	Spiral DAMS Strategy
Description	Evolutionary (multi-pass) and concurrent (with overlaps of activities between and within iterations)
Sequential or Concurrent Allocation of DAMS activities	Partition CDS Ontological entities into three contiguous sets to be delivered iteratively
	First iteration to perform all DAMS Program Activities: MSA_1, TD_1, $E\&MD_1$, $P\&D_1$, and $O\&S_1$. for the first set of CDS ontological entities
	Second incremental iteration to perform TD_2, $E\&MD_2$, $P\&D_2$, $O\&S_2$ for the second set of CDS ontological entities. TD_2 to kickstart during $E\&MD_1$ and to be executed concurrently. (Refer to Figure 25)
	Third incremental iteration to perform TD_3, $E\&MD_3$, $P\&D_3$, $O\&S_3$ for the second set of CDS ontological entities. TD_3 to kickstart during $E\&MD_2$ and to be executed concurrently.
	Within-iteration concurrent execution of Program Activities that match those specified in Figure 26.
	Additional risk-based Program Activities to be added. Even though typical DAMS Program Activities have included risk-based reviews, in order to differentiate Spiral from the other two SEP models, the Spiral model should exhibit a greater focus on their characteristic risk-based reviews.
Behavior	The Spiral Model is a process generator and may choose to execute a particular iteration using another SEP model when the context is suitable.
	Ordered, but concurrent flow through the model.
	Detected emergence can be factored into next increment's TD.

F. INPUT, CONTROL, OUTPUT VARIABLES

1. Input Variables

The following describes the list of User defined input parameters to the CDS Simulator (CDSS).

- Initial State. A set of parameters representing the initial capability needs, described by the Capability, its set of Functions, associated Requirements, associated Emergent Traits, performing Components and responsible Organizations at the start of simulation.

- Final State. A set of parameters representing the as-deployed emergent traits, described by the Capability, its set of Functions, associated Requirements, associated Emergent Traits, performing Components and responsible Organizations at the start of simulation.
- Transitive States. A set of parameters representing the insertion of capability needs, described by (1) new Components, with their set of Functions, associated Requirements and Emergent Traits and (2) existing Components, with their set of updated Functions, associated Requirements and Emergent Traits. The set of parameters is tagged with a numerical value corresponding to a simulation time during which they would be inserted.
- Choice of SEP models. A list of SEP models to be loaded for the simulation.

2. Output Variables

Conventional wisdom posits that there are gaps between capability performance articulated as-needed (i.e., what the joint staff wanted), the performance based on system architecture as-planned (i.e., what the lead DoD component planned), and the capability of an SoI as-deployed (i.e., what the end user got and how the end user actually uses the capability). The CDS ontology with Fog of Emergence allows us to add another set of attributes to a CDS, the capability performance as-known (i.e., what we know the users would get and how they use the capability). The four sets of attributes for the CDS are as elaborated below:

- The as-needed attributes are measured off the set of requirements associated with capability needs for its Intended Missions from its initial state towards the end of the capability's life cycle with changes due to JCIDS insertion of needs, changing face of war, politics or appearance of disruptive technologies.
- The as-planned attributes are measured as a proper subset of the as-needed attributes that a Lead DoD component has recognized and planned for delivery through the CDS.
- The as-deployed attributes are simply the summation of the full set of emergent traits (beyond those desired as requirements) exhibited by the functions performed by the SoI components in **all** Mission Contexts.

- The as-known attributes are measured off the organizations' subjective knowledge of the as-deployed attributes as perceived through the Fog of Emergence. The capability delivery organizations as well as end-users could only make use of as-known attributes in their decision making and plans with respect to capability delivery. The as-known set of attributes can only be grown through successful analysis of how the SoI would work for successful detection of the emergent traits exhibited by the SoI in its operational contexts (which is a subset of all Mission Contexts but more than the Intended Mission). The as-known is always less than the as-deployed.

It should also be said that the as-deployed set of attributes is a theoretical construct as it is predicated on omniscience regarding the set of Emergent Traits associated with the SoI in all Mission Contexts. However, this as-deployed set of attributes is meaningful in the context of this thesis as a common benchmark to examine the CDS performance measure through attributes as-needed, as-planned, and as-known. The most important contributions of this thesis are: (1) to provide a CDS ontology that could act as a prism to separate the white light of capability performance into its constituent colors of "as needed," "as-planned," "as-known" and "as-deployed;" and (2) to gather insights into how capability delivery Organizations could try to expand their "as-known" perspective as much as practicable.

3. Control Variables

The following are the variables that the User of the CDSS could control to explore the effects on the Measures of Interest.

- SEP models provide different strategies toward allocating the DAMS life cycle phases and toward clearing the Fog of Emergence.
- Variability of capability needs could be directly manipulated through the number of capability needs inserted into the CDSS during simulation.
- Complexity of capability implementation could be indirectly manipulated by varying the number of dummy work packages to be inserted into the work space. A highly complex SoI is likely to result in greater rework, as a consequence of relatively lower requirement engineering, SE, and domain-specific engineering competencies with respect to the inherent complexity of the SoI

G. IMPETUS FOR EXPLORING CAPABILITY DELIVERY SYSTEM SIMULATOR

As part of the research approach, a conceptual Capability Delivery System Simulator (CDSS) and its critical functionalities would be explored. There are four reasons behind the inclusion of the CDSS.

First, a deeper understanding on the tractability and shortcomings of the proposed CDS ontology with fog of emergence can be gained from conceptualizing and exploring the development of critical CDSS functionalities.

Second, the encoded logic within the exploratory CDSS would serve as a less ambiguous extension of the prose captured in the main paper of this thesis and facilitate further discussion and improvement of the body of knowledge regarding capability delivery with emergence.

Third, the measures of effectiveness for capability delivery, as particular regards to the “as-deployed” attributes, are theoretical constructs that are best discussed with assistance of a simulator that is built on the CDS ontology with fog of emergence. This exploratory CDSS is the first step toward developing a full-fledged simulator system.

Finally, the concepts and software functionalities developed could be also applied into other capability delivery software applications apart from simulation.

In the following chapters, we would elaborate on the conceptualization of the CDSS requirements, functionalities and preliminary design before a summary of the insights gained from the implementation of key CDSS functionalities.

IV. CDSS CONCEPT

As the CDSS is a simulation model based on the CDS ontology with emergence, which in turn is extended from Vitech's CORE 8 ontology, the CDSS might be implemented in Vitech CORE. While the subsequent exploratory implementation of the CDSS was in JAVA programming language, the CDSS concepts captured in this chapter would be implementation agnostic in nature.

A. PURPOSE

The purpose of the CDSS is to model the CDS to explore the effects of SE process models on how the CDS would adaptively change its as-needed, as-planned and as-known capability needs/requirements given the same as-deployed capability and noise factors.

The CDS would take in a User defined as-deployed capability and a set of initial capability needs. The noise factors would include instability of capability needs provided as capability needs that would be inserted at User-defined intervals. These inputs are analogous to what the JCIDS would be subjecting the CDS.

While the CDSS's primary purpose is to measure the as-needed, as-planned and as-known capability needs/requirements, the CDSS should also provide some means to determine the cost and time-taken as the capability proceeds through its life cycle toward deployment.

The measures of effectiveness for the CDS would be to minimize gaps between as-needed, as-planned, and as-known. Figure 26 shows how the capability needs for the capability could change over time and values "x" and "y" represent the gaps. The as-known line in Figure 26 is lower than the rest, due to negative emergent traits that decreased the achieved performance of a requirement from what was planned. The opposite could happen in the case that emergent traits are known to help raise achieved performance of requirements.

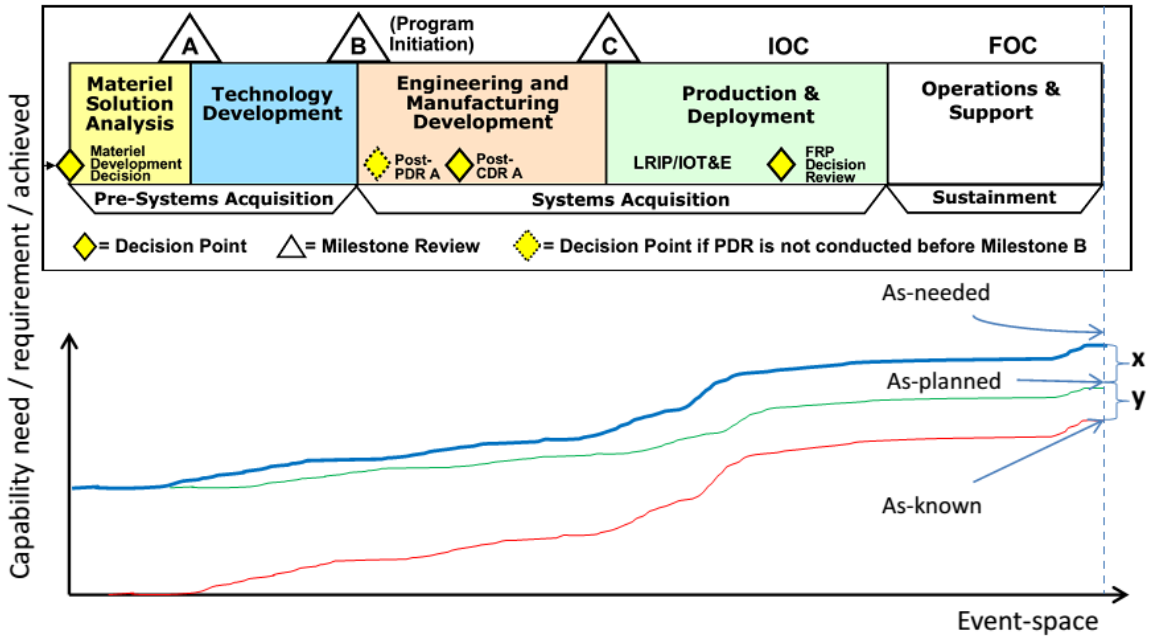


Figure 26. Different types of capability needs over the SoI's life cycle (After OUSD AT&L, 2008).

B. SCOPE

The scope for the CDSS is shown in Figure 19; specifically it excludes modeling of the JCIDS. The inputs and noise factors from the JCIDS would be made available to the CDSS as user-defined data.

The choice of SE process models and the maturing SoI over the DAMS life cycle phases as supplied by the capability delivery organizations are within scope.

C. REQUIREMENTS LIST

The following section captures the list of software requirements derived from the conceptual model and would be refined iteratively along with the spiral. They are to be read in totality along with the requirement models that follow in Section 4 to gain a holistic understanding of the requirements.

1) The CDSS shall model system life cycle phases as Program Activities based on the Defense Acquisition Management System.

a) The CDSS shall initialize workspaces (comprising of Program Elements) for each Program Activity and allocate them to Organizations responsible for them

b) A Program Element is a representation of work that when performed would contribute to the supply of an operational architecture artifact, a system architecture artifact, or a SoI component.

c) A workspace is a collection of Program Elements under the same Program Activity.

d) The CDSS shall use these Program Activities as the list of discrete events to drive an event-based simulator.

2) The CDSS shall allow the User to specify a Systems Engineering Process (SEP) model to be loaded for the simulation (refer to Chapter III).

3) The CDSS shall adapt the initialization of the phased work packages and events according to the selected SEP models as documented in Chapter III.

4) The CDSS shall implement the CDS ontology model as described in Chapter IV Section. D.2 of this chapter to model how a capability is matured over its life cycle due to SEP model-specific interactions amongst the three domains of operational architecture (capability needs), system architecture (capability implementing system with emergent traits), and program management (work packages and events) across the life cycle phases.

a) The CDSS shall allow the User to specify two sets of capabilities performances for an SoI:

i) As deployed capabilities: The full set of capability needs, associated SoI, and all emergent traits as represented in the ontological format from Figure 21.

ii) Initial capability needs: A subset of the capability needs, parts of SoI and emergent traits representing what is initially known to the capability delivery organizations at the start of simulation.

b) The CDSS shall allow the User to specify the values for the following attributes based on the data table in Table 12:

i) Capability Delivery Organization: A User defined list of engineering competencies and the level of competency for each competency possessed by the organization.

ii) Emergent Trait: A User defined set of engineering competencies and level of competency needed to analyze or measure this trait.

c) The CDSS shall allow the User to specify the simulation time during which the difference in capability needs and SoI between the initial and as-deployed capabilities would be inserted into the CDS as noise factors:

i) Consistency of specified time across different runs: The CDSS shall allow the User to specify in a consistent manner as to when a particular capability need and associated parts of SoI is to be inserted during simulation time.

ii) Consistency refers to the ability to specify this insertion at the same unit of simulation time across different runs involving the same sets of capabilities as defined in Requirement 4.a. The consistency across runs ensures a common base of comparison across SEP models.

d) The CDSS shall be able to create random workspaces filled with the appropriate and work packages for each life cycle phase satisfying the following sub-requirements:

i) The randomly generated workspaces shall be validated against the as-deployed capability needs to ensure that there exist a set of work packages (program elements) that would supply the needed Operational Architecture, System Architecture Artifacts and SoI components that would exhibit as-deployed emergent traits that satisfies the as-deployed capability needs.

ii) When a workspace is validated, a random number of dummy work packages would be randomly inserted into the workspace. These dummy work packages would have the same attributes as the true work packages except that they would fail a verification check. Dummy work packages represent work performed that does not contribute productively towards the supply of any artifacts of SoI components.

5) The CDSS shall be able determine a pair of estimated cost and schedule needed to perform a set of work packages is sufficient to produce the corresponding capability architecture artifact, system

architecture artifact or system component that satisfies the capability needs within a life cycle phase.

a) Consistency of estimated cost and schedule: The CDSS shall apply a repeatable cost and schedule estimation heuristic so that the same cost and schedule would be calculated for different runs involving the same workspace.

b) The heuristic shall be applied on the initial capability needs to determine the estimated time and budget for each life cycle phase.

6) The CDSS shall maintain a time-sorted queue of events comprising of DAMS Program Activities and SEP-model process generated events.

7) The initial DAMS and SEP-model process events shall be inserted into the queue using the time determined in Requirement 5.b.

8) The CDSS shall allow for at least three types of events as listed in the following sub-requirements:

a) DAMS Program Activity events. They have associated work packages that when performed help accomplish these milestones and events.

b) Work Completion events. When the planned work packages could be completed before their associated event, a “work completion event” shall be created and inserted into the time-sorted event queue. This event could potentially be used to keep track of the amount of slack-time from work completion to next event.

c) Capability needs insertion events. These are events that where capability needs are inserted to the CDS at simulation time specified under Requirement 4c.

9) The CDSS shall process the earliest event in the queue with the following basic outcomes:

a) As the simulation is event-driven, the time tagged on the earliest event denotes the effective simulation time. If there is any cumulative schedule slippage (refer to Requirement 9.k), the effective simulation time shall be the sum of the time tagged on the event and the cumulative schedule slippage.

b) The CDSS shall perform a verification check of all work packages performed up till the effective simulation time and reveal any dummy work packages inserted according to Requirement 4.d.ii.

c) The CDSS shall determine how many artifacts and system components have been supplied based on the current set of valid work packages performed.

d) The CDSS shall randomly determine if the corresponding capability delivery organizations would be able to increase their knowledge of the emergent traits and mission contexts associated with the set of completed artifacts and SoI components.

i) The CDSS shall model the possible increase in knowledge of emergent traits according to the state transition diagram shown in Fog of Emergence, Figure 20.

ii) The CDSS shall ensure that organizations could only lift the fog of emergence on mission contexts during Requirement & Analysis sub-Program Activities. This lifting of fog is because organizations responsible for the operational architecture would be more focused on identifying concepts of operations.

iii) The CDSS shall ensure that organizations could only lift the fog of emergence on emergent traits that manifest in intended missions and are expected as requirements during the Design and Architecture sub-Program Activities. This lifting of fog is because of the tendency for design organizations to design to requirements for intended missions.

iv) The CDSS shall ensure that organizations could only lift fog of emergence on emergent traits that manifest in intended missions during Development and Acquisition sub-Program Activities. This lifting of fog is due to the tendency for builder organizations to be only concerned with building a SoI component to work in intended missions.

v) The CDSS shall allow organizations to lift the fog of emergence for mission contexts and for emergent traits that manifest outside of intended missions during Integration and Test sub-Program Activities. This lifting of fog is because integration organizations would be concerned with how the system would be used instead of what it was designed for.

vi) The random mechanism shall take in to account the differences between the required competency level and the competency level possessed by the

organization to adjust the probability of successful analysis or detection.

(1) If the required competency level is lower than the organization's competency level, the probability of success is one.

(2) If the required competency level is higher than the organization's competency level, the probability of success should be reduced.

e) Undesirable emergent traits shall be modeled by assigning a negative value.

f) The achieved performance of a requirement is the summation of all emergent traits that contribute to it.

g) The CDSS shall reveal work packages corresponding to any newly known negative emergent trait that does not affect any desired requirement.

i) Such negative emergent traits represent traits that arise due to the multiplicity of the as-deployed operational context and have a real impact on the performance of the SoI despite being unanticipated.

ii) Even though not associated with any SoI requirement, such negative emergent traits should be ideally reduced to zero by going through due diligence to perform corresponding work packages that would contribute to their suppression.

h) The CDSS shall calculate an organization's latest knowledge emergent traits exhibited by the current set supplied of artifacts and SoI components as viewed through the Fog of Emergence.

i) The CDSS shall determine if the latest subjective knowledge of emergent traits satisfy the latest capability needs.

i) If the traits satisfy the needs and does not have any known negative emergent traits, this event does not need to be rescheduled.

ii) If the traits do not satisfy the needs or have known negative emergent traits, this event has to be rescheduled.

j) If the event does not have to be rescheduled, the CDSS shall proceed to process the next event.

k) If the event has to be rescheduled, the CDSS shall apply the cost and schedule heuristic in Requirement 5 to determine a

new set of work packages to be performed and then to process the event again but with a new effective simulation time based on the new estimated time needed.

l) If the event has to be rescheduled, the CDSS shall keep track of the *cumulative schedule slippage* in simulation time units, to allow the existing events in the queue to be offset by the appropriate amount of simulation time, without a need to update each and every event in the queue.

10) The CDSS shall allow SEP-model strategies to alter the event processing basic outcomes listed in Requirement 9 according to the following sub-requirements. Where conflict arises, this Requirement shall take precedence over Requirement 9. The list of SEP-model strategies can be found in Tables 9 through 11 in Chapter III.

a) Addition to Requirement 9.i: The SEP-model strategy may specify how and when the CDSS would allow capability delivery organizations to receive new capability needs (if any) at that given effective simulation time.

b) Addition to Requirement 9.j: If an event does not have to be rescheduled after being processed, the CDSS shall check to see if this event is a "work completion event." The SEP-model strategy may specify how the CDSS could handle early completion of work ahead of a scheduled event.

c) Addition to Requirement 9.k: If an event has to be rescheduled after being processed, the CDSS may use an SEP-model strategy-specific modification of the cost and schedule heuristics (Requirement 5) to determine a new set of work packages to be performed and whether the resulting updated event could be directly processed or inserted into the time-sorted event queue.

D. REQUIREMENT MODELS

1. Use Cases

Three use cases for the CDSS are listed as shown in Figure 27. The User is expected to "Start simulator" which would draw inputs from an Input File. Thereafter the User could decide either to run in "event trail mode" or "no trail mode" with the former use case constantly writing to the Console to provide a textual update on the simulation progress. Both modes would use the "Run simulator" use case, which in turn prints the final results of the simulation to Console and to an Output File.

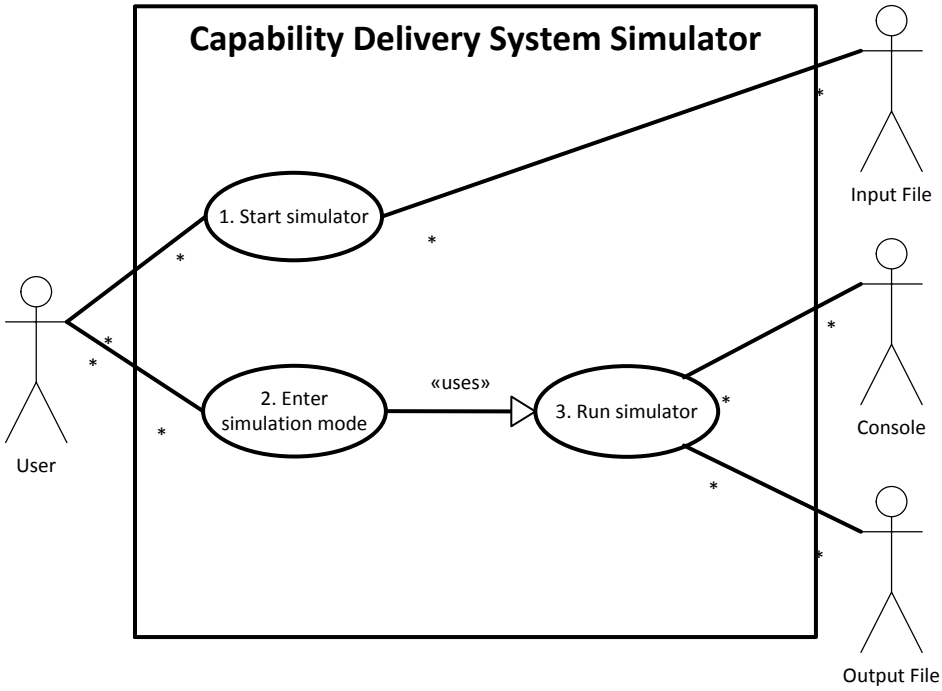


Figure 27. High-level use cases for CDSS.

For the details regarding the use cases, refer to Appendix B.

2. Data Attributes

This section captures the additional data attributes for the CDS ontology with Fog of Emergence shown in Figure 20. For the full set of data attributes, please refer to the *CORE 8 Architecture Definition Guide*.

Table 12. Additional Data attributes for CDS Ontology with Fog of Emergence.

S/N	Attribute	Description
Organization		
1	Competencies	A list of Strings indicating the engineering competencies that the organization possesses.
	Competency levels	Competency levels: A list of numerical values (0–1) indicating the corresponding levels of competency.
Emergent Trait of a Function		
2	(Primary Key) Name	A String that identifies the name of an emergent trait. If an emergent trait is desired as a Requirement, this field has to match the corresponding Name field of the Requirement.
	(Primary Key) Function	A String that identifies the corresponding Function that exhibits this emergent trait.
	Value	A numerical value that can be either negative or positive denoting the contribution of this emergent trait arising from the associated Function.
	Determinable	A numerical value that can be either negative or positive denoting the contribution of this emergent trait arising from the associated Function.
	Requisite engineering competencies	A list of String values indicating the requisite engineering competencies needed to analyze or measure this emergent trait.
	Requisite competency levels	A list of numerical values (0–1) indicating the corresponding levels of competencies needed to analyze or measure this emergent trait.
Fog of Emergence		
3	(Foreign Key) Organization	A String value indicating the name of an Organization.
	(Foreign Key) Emergent Trait	A pair of Emergent Trait-Function String values identifying the indicating the name of an Emergent Trait of Function.
	State	A String value representing the associated Organization's state of knowledge for the associated emergent trait.
	Subjective Knowledge	A numerical value that shows the Organization's extent of knowing the actual contribution for an associated Emergent Trait.
Function		
4	Emergent Traits	A numerical value that shows the Organization's extent of knowing the actual contribution for an associated Emergent Trait.

3. Activity Diagram

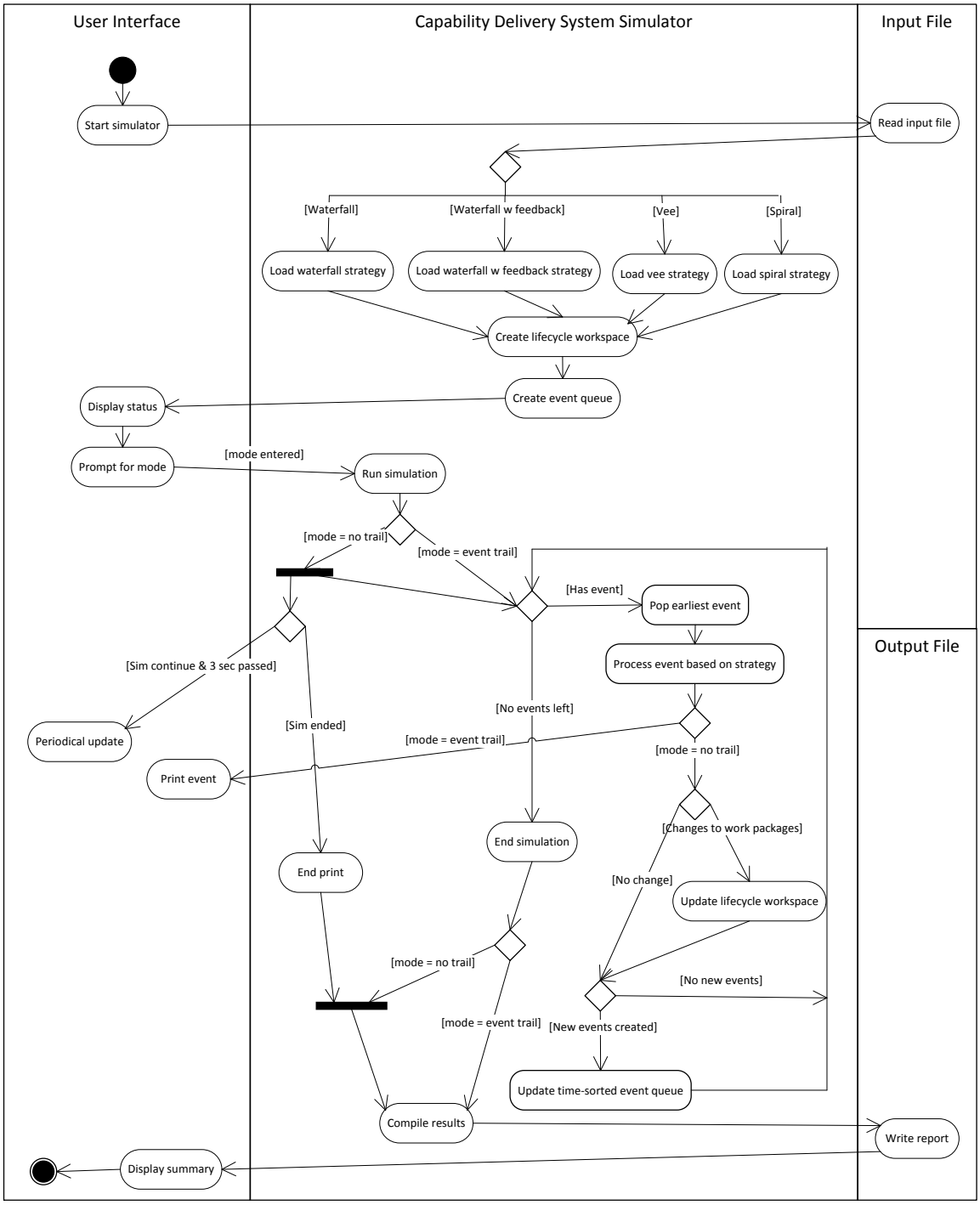


Figure 28. Swimlane activity diagram for the CDSS.

The User starts the simulator using a command through the user interface, and the CDSS reads the corresponding input file to load an appropriate strategy for the SE process model and to initialize its internal representation of the capability life cycle workspace. Using the SE process strategy, a time-sorted event queue based on the DAMS life cycle events and the SE process model events would be created. The CDSS displays a summary status of what it has been initialized with and prompts the user for a simulation mode. The CDSS notes of the selected mode and begins to run the simulation. If the mode is set to “no trail,” a parallel update thread for the user interface is created. The main discrete event processing thread is entered for both cases of “no trail” or “event trail” mode.

The discrete event processing thread on the right side of the diagram picks the earliest event from its queue of time-sorted events and processes the event according to the selected SE process model strategy.

If the mode is set to “event trail,” the simulation thread would print the details of this event to the user interface, allowing the User to be apprised of the current as-needed, as-planned, and as-known capabilities. Depending on the strategies there might be updates due to either the life cycle work packages or creation of new events.

The thread checks the time-sorted event queue for more events to process, and if there is an event to be processed, the event processing sequence is repeated. This discrete event processing thread transitions to end simulation when there are no more events to be processed.

The “no trail” user interface update thread runs in parallel to the discrete event simulation thread. The thread simply prints a period to the user interface every three seconds as a visual indicator to the User that the discrete event processing thread is still running. This user interface update thread transitions to end print when the discrete event processing thread ends.

If a “no trail” user interface user interface update thread was created, this update thread would rejoin the main discrete event processing thread. The CDSS then compiles the results of the simulated run and writes a full report to the output file before printing a summary to the user interface and exiting.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CDSS PRELIMINARY SOFTWARE DESIGN

A. CDSS ARCHITECTURE

The CDSS is a software-intensive system developed using the Java Platform, Standard Edition 7 without using any Operating System (OS) specific libraries. The Java Platform provides a Java Runtime Environment that abstracts the underlying hardware and OS away from the Developer. As long as the User obtains the appropriate Java Runtime Environment[12] for their computers, the Java Platform would execute the CDSS in the same manner across different hardware and OSs.

The following section articulates the functional and component architectures that illustrate all requisite functionality have been implemented.

1. Functional Architecture

Figure 29 shows the CDSS Functional Architecture, up to four-levels of decomposition of the functions required to implement the requirements of the CDSS. The leaf functions have been numbered 1 to 21.

The top level functions are:

- Read input: To read the external input file into the CDSS.
- Prepare simulation: To prepare the simulation according to the inputs received.
- Run simulation: To run the discrete event simulation.
- Record data: To record data to memory.
- Write output: To write data to Excel.

12 Please access http://java.com/en/download/index.jsp to download the latest version of the Java Runtime Environment appropriate for your hardware and OS.

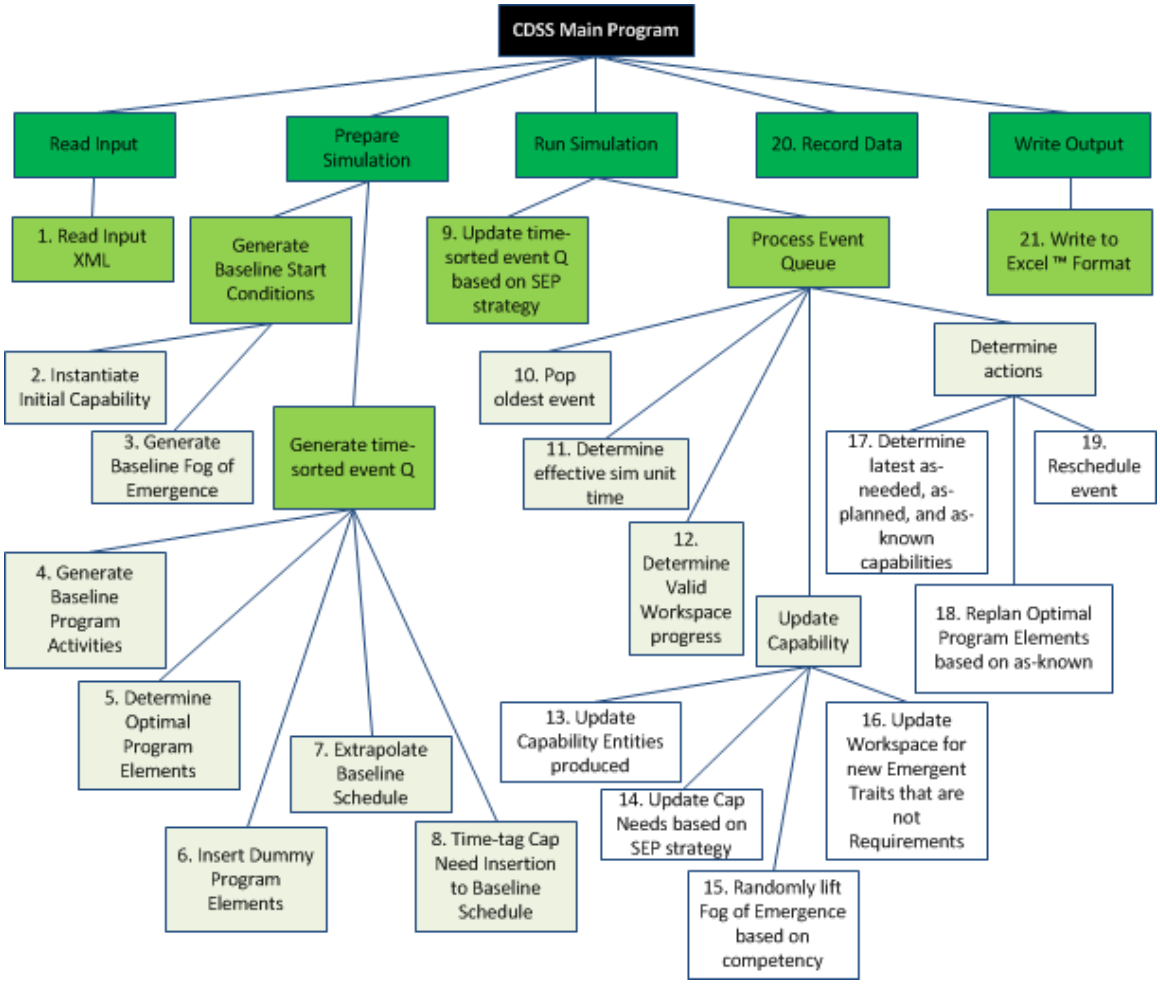


Figure 29. CDSS Functional Architecture

2. Component Architecture

The Component Architecture allocated with the requisite functions is shown in Figure 30. The CDSS Executive uses the Initializer, Runner, and Util components to initialize the simulation before running the simulation, supported by the utilities of an XML reader, data recorder, and Excel writer.

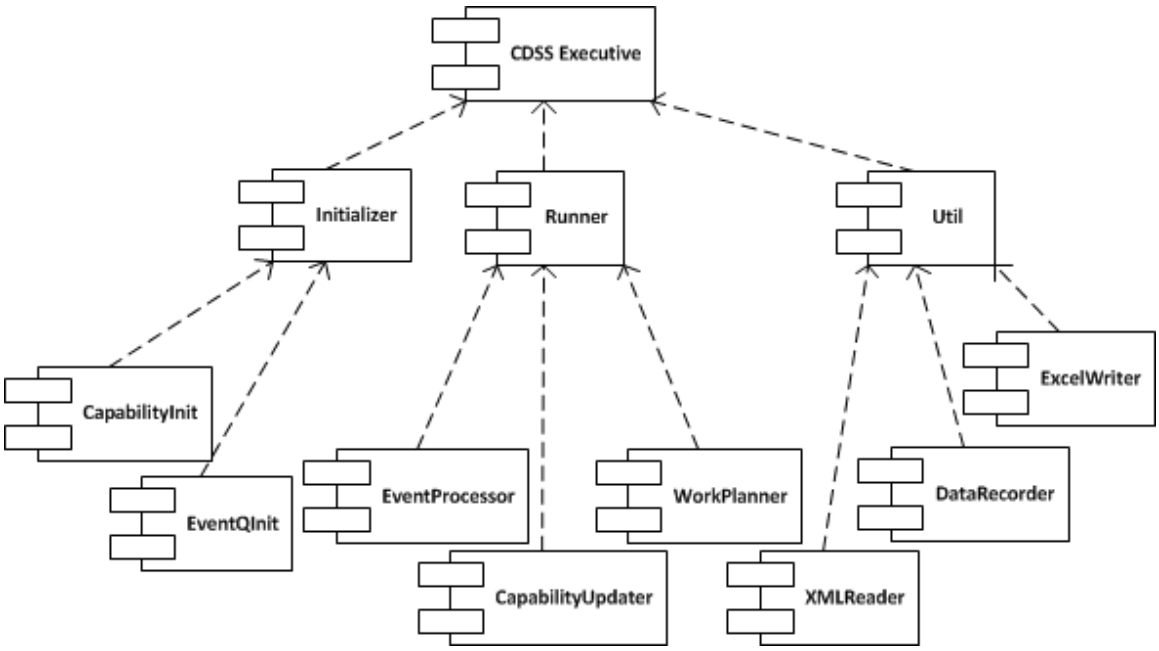


Figure 30. CDSS Component Architecture

The allocation of functions to components was derived from performing a N2 chart analysis of the functions, and then grouping the functions through the heuristics of increasing module-internal cohesion and decreasing module-external coupling.

As shown in Figure 31, functions 2 through 8 (annotated leaf functions on page 88), form the Initializer component; functions 9 through 19 make up the Runner component; and lastly functions 1, 20, and 21 form the Util component.

The Initializer component is broken into the CapabilityInit and EventQInit components responsible for creating the baseline Capability Architecture and time-sorted event queues respectively.

The Runner component is broken into three sub components, with function 9 (queue updating based on SEP strategy) being kept at the parent component-level as function 9 have coupling with functions from two sub-components. The EventProcessor component processes the oldest event, the CapabilityUpdater component updates the Capability Architecture, and the

WorkPlanner component take stock of progress and updates work plans if required.

The Util component comprise of three sub-components, the XmlReader, the DataRecorder, and ExcelWriter. These sub-components are highly coupled with the other components, and hence were kept as an external high-level component to be re-used instead of redundantly duplicated.

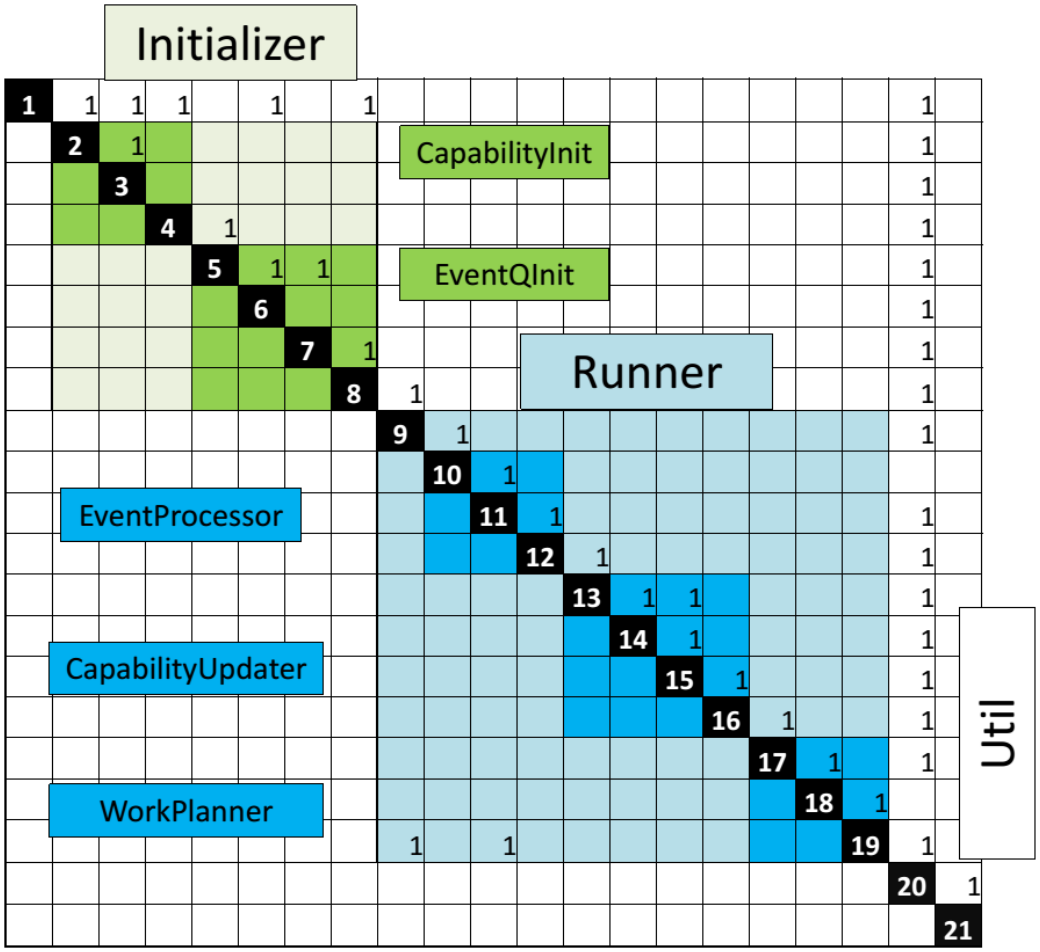


Figure 31. N^2 Chart of CDSS Functions and Component Grouping[13]

[13] A "1" in a cell of x-th row and y-th column denotes information flow from the corresponding function on the x-th row to a function on the y-th column.

VI. EXPLORATORY CDSS IMPLEMENTATION RESULTS

A. SIMPLIFYING ASSUMPTIONS

Although the exploratory CDSS developed as part of this thesis is not fully functional for the purpose of running experiments, it current form is capable of a complete normative simulation run.

The normative simulation run takes a specifiable capability need through all DAMS life cycle activities of MSA, TD, E&MD, P&D, and O&S, driven by a class Waterfall SE process model, subjected to unstable capability needs that change during simulation. The fog of emergence for a single capability delivery organization was implemented and could be dynamically lifted at run-time to reconcile the organization's perceived "as-needed," "as-planned" and "as-deployed" attributes of a capability as it steps through the capability delivery life cycle.

While, the exploratory implementation CDSS's is facilitated by many simplifying assumptions the underlying software wireframe maintains the ontological relationships between the CDS ontological entities according to the proposed CDS ontology with Fog of Emergence. In this section we cover the key simplifying assumptions. The discussion in the main paper of the thesis shall use prose and diagrams as much as possible but code snippets will be included where useful.

1. One-to-one Perfect Match Between Associated CDS Ontological Entities

In typical systems engineering, the Operational Activities to achieve Intended Missions of an Operational Architecture are represented as operational flow blocks that are further refined into Functional flow blocks and Component hierarchies in the System Architecture. While systems architecting heuristics advise a one-to-one allocation of function to form, in practice, it is often unrealistic to ensure that a component is only allocated a single function.

In this thesis, however, we apply this heuristic and assume a perfect match for associations between CDS ontological entities of the same architecture type and across architecture types as shown in Figure 32.

For CDS ontological entities such as the Performer, Capability, and Operational Activity, they would refer to the same set architecture flow blocks. For CDS ontological entities such as the Component, Requirement, and Function, they would refer to the same set of hierarchical blocks.

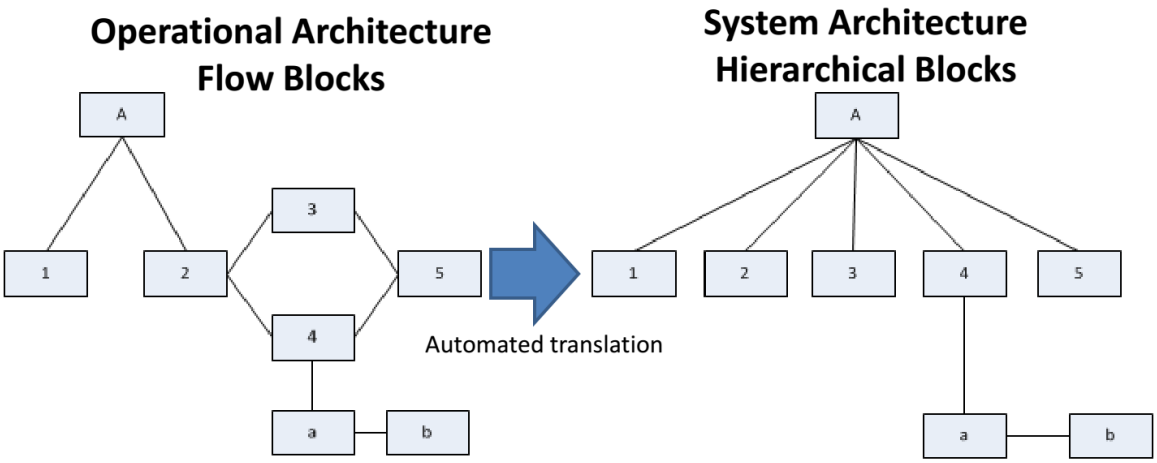


Figure 32. One-to-one perfect match between Operational and System Architecture Entities

The result of simplification facilitated: (1) greater ease for user input (as we only need to specify two sets of blocks for the whole CDS ontology); (2) automated translation of Operational Architecture entities into their corresponding System Architecture entities; and (3) reduces computing resource usage when many CDS entities share the same reference to either a recursively decomposable network of flow blocks or hierarchical blocks.

2. Strictly Sequential Execution of Program Activities

The CDSS is a discrete event simulation, which queues CDS events such as Program Activity, work completion and capability need insertion using a time-

tag determined using a time-allocation heuristic based on perfect knowledge of all as-deployed Capability needs, Emergent Traits and Program Elements.

Intuitively, we know that perfect capability delivery is impossible and we expect a capability delivery Organization that is obscured by the Fog of Emergence to take a longer time to deliver the same Capability when everything else is held equal. During the simulation, Program Activities are expected to be delayed as Organizations encounter rework or choose a less a non-optimal plan of Program Elements to accomplish the Program Activities.

This means that the CDSS has to dynamically recalculate the time tags for CDS events when schedule slippages occur. This was circumvented by keeping track of a single cumulative schedule slippage value. Instead of using CPU cycles to recalculate the time tags for all CDS events in the queue every time a schedule slippage occurs, we simply determine that the effective simulation time is the sum of the original time-tag and this cumulative schedule slippage value.

Unfortunately, even though the exploratory implementation of Program Activities supports recursive and dynamic flow composition, the work-around prevented us from leveraging on that behavior. Referring to Figure 33, assuming we have four Program Activities labeled 1,2,3, and 4 that each activity is expected to 1,2,3, and 4 days to complete respectively. A delay by one day for activity 2 would impact the completion dates of all activities that follow it in the sequential activity flow, but not for the concurrent activity flow where activity 2 is not on the critical path. A mechanism that only tracks a single cumulative schedule slippage would not be able to help determine the schedule impact to non-sequential ordered Program Activities.

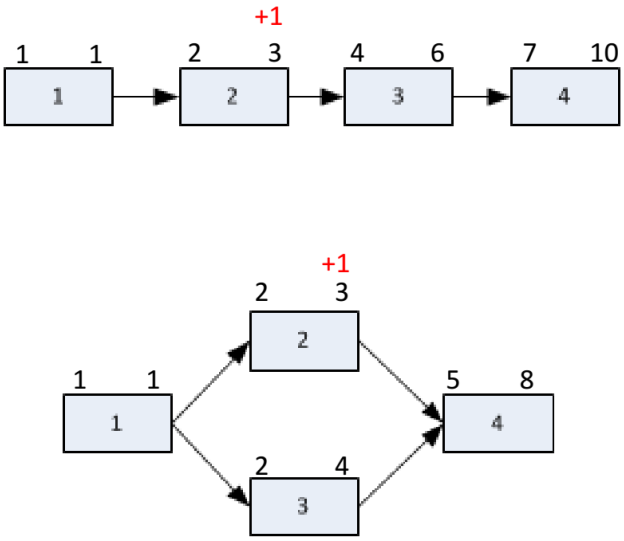


Figure 33. Impact of delays for Program Activities

3. Conflation of the "As-Planned" and "As-Known" Perspectives

The CDS ontology and the Fog of Emergence facilitate the discussion of the divergence between what was "as-planned" and "as-known" by the capability delivery Organizations. However, in the implementation of the CDSS, it was assumed that when not subjected to constraints in maximum time and cost for the overall life cycle, the capability delivery Organization would always plan based on what they know.

Future work could incorporate the concept of Risk into the CDSS to model how capability delivery Organizations might choose to plan differently from what they know.

4. Planning and Execution of Program Elements

Referring to Figure 22 in Chapter III, each sub-Program Activity such as the Materiel Development Decision, Analysis of Alternatives, or Initial Operational Test & Evaluation comes with their own set of Program Elements to evolve the CDS ontological entities. The planning and execution of Program Elements to accomplish Program Activities are obscured by the Fog of Emergence, namely due to a capability delivery Organization's propensity to focus on intended

missions and requirements when delivering a capability instead of looking beyond that to uncover more mission context and emergent traits the as-deployed capability would deal with. Furthermore, the Organization is also subjected to unstable capability needs that may be allocated to the SoI during its life cycle. These factors combine to make the planning and execution of Program Elements an interesting challenge to be modeled.

In the exploratory implementation of the CDSS, each cell in Figure 22 comes with a work space of Program Elements. The workspace is a 2-dimensional box filled with randomly located Program Elements that would contribute towards producing a deliverable that helps the accomplishment of the associated Program Activity as shown in Figure 34. When the Organization lifts more of the Fog of Emergence during simulation time, the accompanying Program Elements associated with the newly discovered Mission contexts or Emergent Traits would also be revealed. Furthermore, to model the relative complexity of the workspace for the Organization, there is a chance that some of the Program Elements in the workspace are unproductive. That is when an unproductive Program Element is performed; it does not contribute toward the accomplishment of the deliverables of the Program Activity.

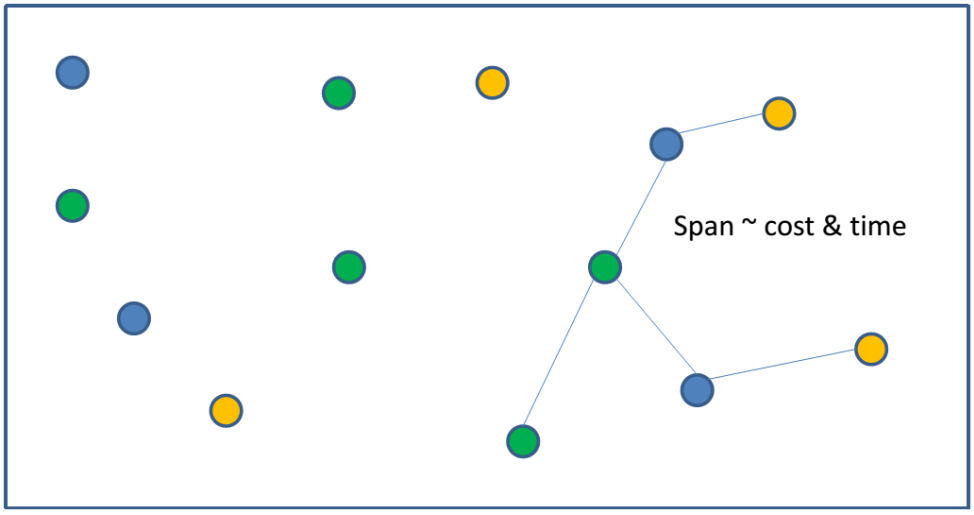


Figure 34. A Program Activity's workspace comprised of Program Elements that help accomplish it.

This representation facilitates the modeling of an Organization's optimizing behavior toward planning and execution of Program Elements where they are expected to try and do enough using the least time and money. To be more specific, the Organization's optimizing behavior was implemented using an analogy of the minimum spanning tree, which is a tree that covers a required set of nodes with the shortest span of the edges. The span of the edges corresponds to the time and money expended by the Organization to perform that Program Element.

In this example, Program Activity requires the production of three deliverables (green, blue, and orange) and each of the deliverables require two Program Elements to be performed. Figure 34 shows a minimum spanning tree that satisfies these conditions. When an Organization performs the planned Program Elements, they would be able to know if the Program Element was productive or not. Figure 35 shows a scenario where one of the planned Program Element turned out to be unproductive (white node), triggering the Organization to incur more time and money (red edge) to rework a needed Program Element.

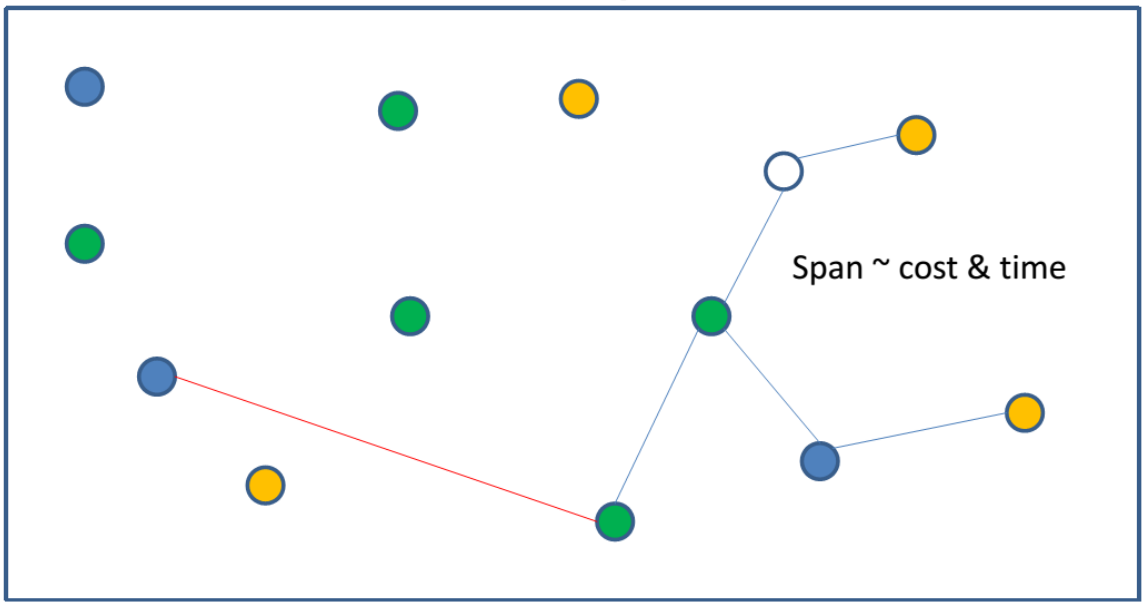


Figure 35. Unproductive Program Elements result in rework

The advantage of such a model is that there is no inherent bias towards any particular SEP model. A SEP model that delivers the capability in increments may partition the workspace into standalone workspaces and tackle them incrementally. Another SEP model that delivers a capability in a single-pass might try to tackle the whole workspace at once.

Conventional Graph Theory implies that a SEP model takes into account the whole workspace at once, which always yields a better minimum spanning tree. However, in our model, both the incremental or single-pass SEP models would be equally obscured by the Fog of Emergence, **initially**. This obscuration means the single-pass SEP model might be expending more cost and time to lift the Fog of Emergence than an incremental SEP model that focused on Program Elements producing a subset of the deliverables. Hence, incremental SEP model has a chance of lifting the Fog of Emergence more rapidly.

5. Aggregated Handling of Entity Attributes

The proposed CDS ontology with Fog of Emergence specifies that each Organization has their own unique set of engineering competencies and competency levels, and each Emergent Trait has their own corresponding requisite engineering competency level. This means that an Organization might be good at analyzing or measuring a particular Emergent Trait while being relatively poorer at doing the same for another Emergent Trait.

However, in our exploratory implementation, we use a single competency level and a single requisite engineering competency level for the Organizations and Emergent Traits. This means that in the normative CDSS run, when an Organization is given the chance to analyze or measure Emergent Traits, the Organization would have same probability of success across all Emergent Traits. The Table 13 captures the use of aggregated handling of attributes that should have been specified and handled individually.

Table 13. Use of Aggregated Handling in Exploratory CDSS Implementation

S/N	Aggregated handling	Implication and Mitigation
1	Single competency value and requisite competency levels to lift the Fog of Emergence, instead of setting corresponding values for each Emergent Trait and each Organization	This means the Organization would always be equally competent in analyzing and detecting all Mission Contexts and Emergent Traits within a single simulation run.
2	Single Requirement threshold value applied to all Functions	The manifested Emergent Traits for a Function in a particular Mission Context is set individually. This means the CDSS still has the level of resolution to differentiate SEP models ' ability to lift the Fog of Emergence between runs.
3	Single complicatedness and complexity factor applied to CDS ontological entities of the same Architecture type. For example, Capability, Performer, and Operational Activity fall under the Operational Architecture while Requirement, Component and Function fall under System Architecture.	These values are used to generate the corresponding DAMS sub-Program Activity workspaces. A more complicated workspace is one with a bigger breathe, allowing a greater dispersion of Program Elements in the workspace and hence possibly causing the Organizations more time and money to do the same Program Elements than in a less complicated workspace. A more complex workspace is one with more unproductive Program Elements, representing the Organization's inability to have a proper grasp on the workspace resulting in more probable rework. This means all Program Activities are equally complicated and complex if they deal with the same Architecture type. This should not be the case as a Technological and Prototype Demonstration Program Activity that deals with the building of Components to the Systems Architecture should not be as complicated and complex as the Follow-on Integration Test & Evaluation of a produced SoI.

6. No Penalty for Doing Work Out of Phase

As an Organization lifts the Fog of Emergence or is subjected to Capability needs inserted into the CDS, it might be triggered to perform Program Elements associated with a Program Activity that has already been passed. Currently there is no additional penalty for doing work out of phase. As a recommendation

for future work, the appropriate DAMS life cycle phase for this capability need could be inserted according to the DoD 5000.02 directive. A penalty function would need to be developed.

B. DISCUSSION OF KEY FEATURES OF IMPLEMENTATION

This section discusses the result of the exploratory implementation of the CDSS using the functional architecture described in Chapter V. The top level functions remain the same, but as not all functions were fully explored some sub-functional headings were renamed where appropriate to better reflect what was done.

1. Read Input

While the User does not have the ability to specify user inputs in text or XML format yet, the current CDSS does provide application programming interfaces into passing important User inputs to the CDSS.

The two sets of User specified inputs to the CDSS are (1) Initial Capability need and as-deployed SoI with the full Set of Mission and Emergent Traits, and (2) when Capability Needs are inserted into the CDSS (if there is a difference between the Initial Capability need and as-deployed SoI). The application programming interfaces to set these parameters are elaborated in the next section.

2. Prepare Simulation

Before the simulation is performed, a number of functions must be performed to set up the simulation. The following subsections would discuss what was implemented to: (a) translate User specified start (Capability need) and end state (as-deployed SoI with full mission sets and Emergent Traits); (b) how to insert transitive states (Capability need insertion) consistently across runs of the same capability but using different SEP models; (c) generate baseline

Program Activities and workspaces; (d) determine optimal life cycle workplan; and (e) how to allocate Program Activities given the strategies of SEP models.

a. Initial Capability Need and As-deployed SoI

In order to specify an as-deployed Operational Activity for a Capability need similar to that in Figure 33, the code to do that is as listed below (with redaction). After instantiating the individual flow blocks, the flow relationship between the blocks are easily set up by calling a function to add the parent-child relationship or a precedent and antecedent relationship. A sample of the software code is shown to illustrate the simplicity of setting up the relationship through the Application Programming Interface.

```
// Setting up the as-deployed Operational
OperationalArchitecture asDeployedOa = new OperationalArchitecture("A",
                              10.0, 40.0, 0.1);

// Setting up flow blocks referenced by the CDS Operational Architecture
// entities
OperationalArchitectureFlowBlock A = new OperationalArchitectureFlowBlock(
                              "A");
OperationalArchitectureFlowBlock one = new OperationalArchitectureFlowBlock(
                              "1");

...
OperationalArchitectureFlowBlock a = new OperationalArchitectureFlowBlock(
                              "a");
OperationalArchitectureFlowBlock b = new OperationalArchitectureFlowBlock(
                              "b");

A.addChild(one);
...
a.addFollowingBlock(b);
...
// Auto translate from flow blocks to hierarchical blocks
SystemArchitectureHiBlock asDeployedHiBlock = FunctionFlow2HierarchyMapper
                                    .translate(A);
// Setting up the as-deployed System Architecture
asDeployedSa = new SystemsArchitecture("A",
                              asDeployedOa.getCapabilityThreshold(), 80.0, 0.2,
                                          asDeployedHiBlock);
// Setting up the list of Missions that the SoI has to achieve
ArrayList<Mission> fullMissions = new ArrayList<Mission>();
fullMissions.add("M1");
fullMissions.add("M2");

asDeployedOa.setMissions(fullMissions);
```

```
// Setting up the Fog of Emergence with what is the end state (asDeplyedSa,
// fullMissions) and the initial state (asPlannedSa)
FogOfEmergence foe = new FogOfEmergence(
                              asDeployedSa.getTopLevelHiBlock(),
                              asPlannedSa.getTopLevelHiBlock(), fullMissions,
                              initialMissions, 0.5, 0.8,
                                     asDeployedSa.getRequirementThreshold());

// Setting objective and subjective Emergent Traits that manifest in Mission
// contexts, for a set of Functions
foe.setObjectiveEmergentTraits("M1", "Requirements", "All", 0.0);
foe.setObjectiveEmergentTraits("M1", "Requirements", "1,2", -2.0);
foe.setObjectiveEmergentTraits("M2", "Requirements", "All", -1.0);
foe.setObjectiveEmergentTraits("M2", "X", "1", -4.0);
```

The code in the snippet above shows how a flow block network is translated into hierarchical structure through a helper class that performs the translation. Only the top-level flow block has to be specified, as the helper class would navigate through the parent-children and precedent-antecedent relationships automatically translate the whole flow network into a hierarchical structure.

Of interest would be how the Emergent Traits are set into the Fog of Emergence through the setObjectiveEmergentTraits method that accepts the following four parameters:

- Mission name: For example “M1” and “M2”
- Emergent Trait type: “Requirements” imply that the Emergent Trait is expected as a Requirement and would take the same name as the Requirement. Anything else, such as “X” imply that the Emergent Trait is not desired as a Requirement.
- Applicable Functions: “All” imply that this Trait manifest in all functions and anything else implies that this Trait manifests only in functions listed such as “1, 2.” Note that “All” is usually used with Emergent Traits expected as Requirements, as by definition all Requirements of Functions are desired Emergent Traits.
- Manifestation delta: A numerical value that denotes how this Emergent Trait would manifest with respect to a threshold value. If this is a Requirement, this delta value would be applied to a Requirement threshold. If this is not a Requirement, this delta value represents a negative Emergent Trait that has to be suppressed by bringing its value up to zero.

Another point of interest would be the setting of a pair of values representing the complicatedness and complexity associated with the Operational Architectures and System Architecture. As mentioned earlier under the discussion on how we aggregated the handling of CDS entity attributes, this was a quick work around to generate workspaces that work on the associated Operational and System Architectures instead of setting a unique pair of values for each workspace.

b. Capability Need Insertion

The exploratory CDSS allows for Capability needs to be injected by specifying the name of the Capability need block (which is the same as the name of the Flow Blocks referenced by all Operational Architecture CDS ontological entities), and a positive numerical value. This value is used to calculate the actual simulation unit time by which the Capability need is made known to the Organization.

```
// ################################################################
// TIME-TAGGED INSERTION OF CAPABILITY NEEDS INTO THE EVENT Q
// ################################################################
// Here we insert cap need "2" and "3" at 0.1 and 0.4 sim time
// respectively.
// The sim time for insertion is calculated using the best solution
// lifecycle. This also ensures that the cap need would be inserted at
// the same time across different runs of the same capability
eventQ.add(new CapabilityNeedInsertion("2", 0.1 * lifecycleSpan));
eventQ.add(new CapabilityNeedInsertion("3", 0.4 * lifecycleSpan));
```

The code snippet above shows how Capability needs “2” and “3” are inserted into the discrete event queue at a simulation unit time of 0.1 and 0.4 of a value called the life cycleSpan. The life cycleSpan value is derived from pre-simulation calculation of how long an omniscient Organization would take to deliver the capability from MSA to O&S. Using this as a common benchmark allows the CDSS to insert the Capability need at the same simulation unit time across runs involving the same capability, but different SEP model.

How the Organization deals with it is dependent on the SEP model employed. If the Organization chooses to take up the new Capability needs, the

corresponding Program Elements would appear in the workspaces for the Organization and hence affect their planning and execution of work.

c. Generation of Baseline Program Activities and Workspace

The Program Activity was implemented as a composite data structure that can have a single parent, many children, precedent and antecedent Program Activities. The flow heuristic follow two simple rules (1) a parent activity is considered complete only when all its children is completed and (2) an antecedent activity could only begin when all its precedent activities are completed. Referring to Figure 37, parent activity A is only considered completed when children activities from 1 through 6 are competed. Antecedent activity 6 could only begin after both precedent activities 4 and 5 are completed. Following this heuristic, both activities 4 and 5 could happen in parallel. Likewise the two separate children branch of {1, 2} could happen in parallel with the children branch of {3, 4, 5, 6}.

The current implementation of the Program Activity could support concurrency but as discussed in Chapter VI.A, the implementation decision regarding how schedule slippages were tracked effectively rendered the concurrency feature of the Program Activity to a backseat. The baseline Program Activities generated in the exploratory implementation of the CDSS is based on the five main DAMS Program Activities (MSA, TD, E&MD, P&D, and O&S) 20 sub-Program Activities (refer to Figure 22) and is set up in a strict sequential manner as shown in Figure 36. This strict sequential behavior is enforced because TD cannot begin until MSA is completed, and MSA is not complete until all its sequential flowing children of MDD, CD, MBSE and AoA is completed.

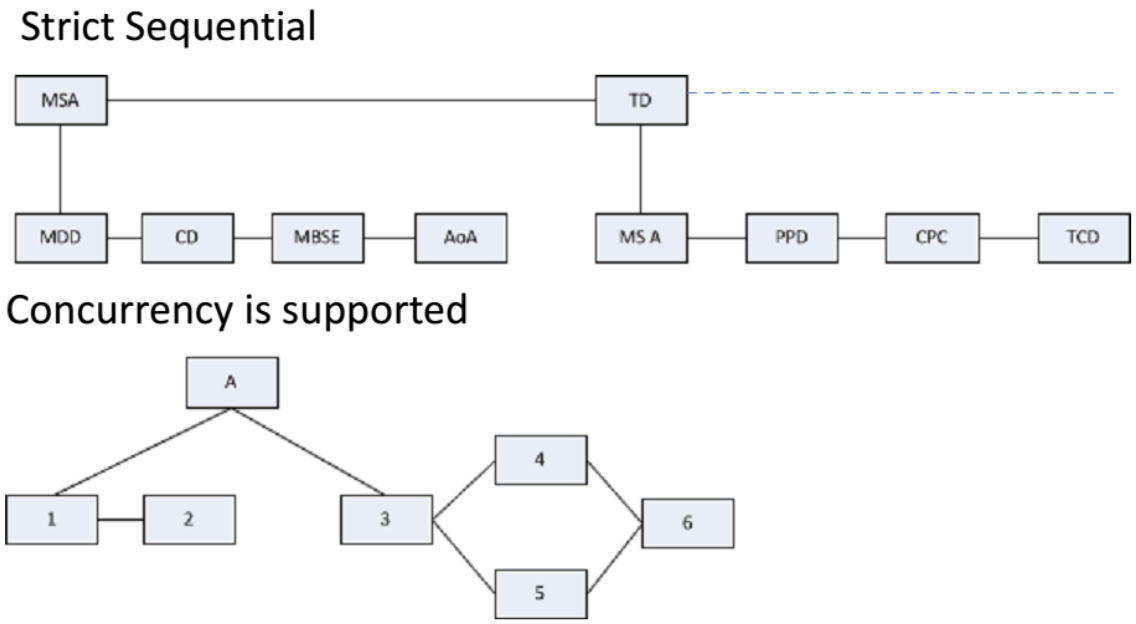


Figure 36. Flexibility in implementation of Program Activity

With the 20 sub-Program Activities set up, the CDSS would then generate the Program Elements needed to accomplish these Program Activities. Figure 37 captures the heuristics employed to generate these workspaces based on User specified inputs. Recall that for each Architecture type, there was an associated pair of complicatedness and complexity numbers. The complicatedness would be used to bound the workspace. For each CDS ontological entity (depicted as Hierarchical blocks in Figure 37) and negative Emergent Trait in the corresponding architecture being evolved by the sub-Program Activity, a number of Program Elements would be generated (shown as nodes with the same color). Program Elements to produce the required entities are shown as circles, while Program Elements to suppress negative Emergent Traits are shown as triangles. In this example, 16 productive Program Elements were randomly scattered in an 80 by 80 workspace. The number of unproductive Program Elements (circles or triangles with dashed borders) to be inserted is calculated using the complexity figure and rounded down to the nearest integer value. The unproductive Program Elements are randomly generated, so it is

possible to end up with all four newly inserted unproductive Program Elements to be for entity 1 or to suppress a single Emergent Trait associated with entity 1.

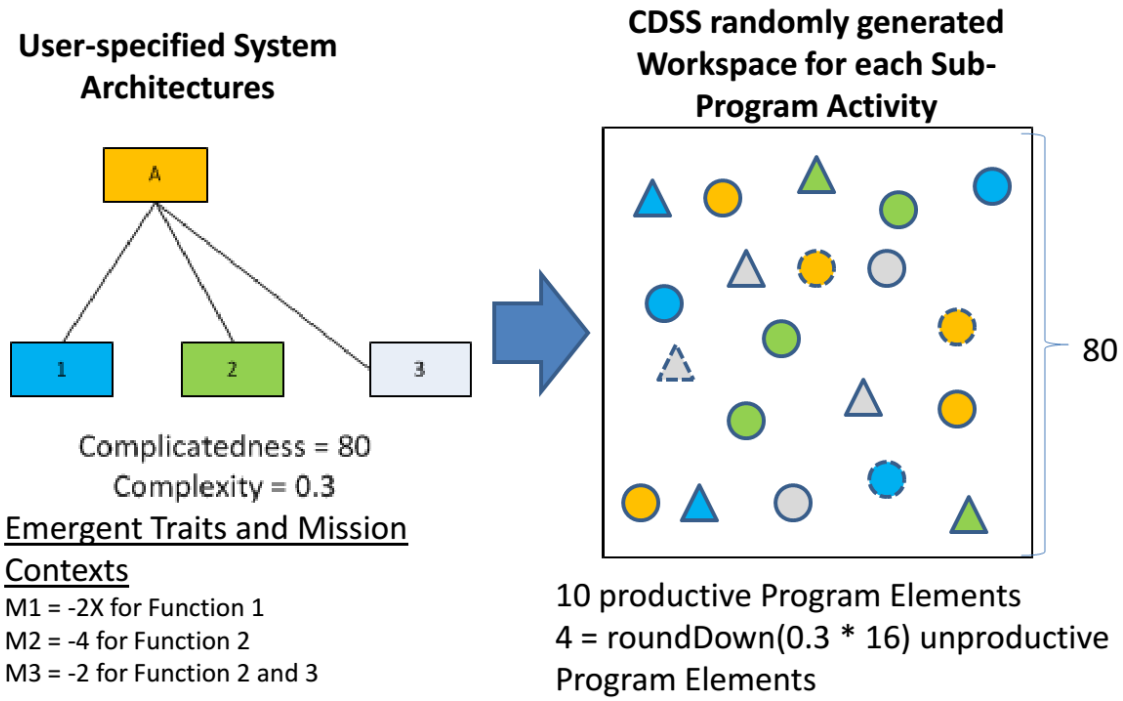


Figure 37. CDSS generated workspace for each sub-Program Activity

It is important to note that in our implementation we ensure that even though the locations of Program Elements were random, no two Program Elements would fall on exactly the same point. This discordance is because we do not want to have an edge that costs the Organization no effort to traverse.

d. Determining an Optimal Life cycle Work Plan

With the 20 Sub-Program Activity workspaces generated, the CDSS then determines the optimal life cycle work plan by following the minimum spanning tree heuristics described in Chapter VI, Section A.3. In this optimal life cycle work plan determination stage, we take the perspective of an Organization that is omniscient and avoids unproductive Program Elements and knows the full

suite of Program Elements associated with the as-deployed Capability needs, Mission contexts and Emergent Traits.

All 20 workspaces would be solved and the respective spans used to set the corresponding planned start and end times for each Program Activity. The total life cycle span is also recorded.

e. Preparation of the CDS Event Queue Based on SEP Model Strategy

The preparation of the CDS discrete event queue for each SEP model strategy is performed after the determination of the optimal life cycle work plan. Unfortunately, the current implementation of the CDSS only covers the Waterfall process model which uses a one to one assignment of optimal life cycle work plan into the CDS event queue. The other strategies as described in Chapter III Section E were not been implemented yet.

At this point, Capability Need Insertion events would be created and inserted into the CDS event queue using the heuristics described in Chapter VI, Section B.2.b.

The final step in the preparation of the CDS Event Queue would be to determine the Organization's work plan based on their subjective knowledge of the Capability as viewed through the Fog of Emergence. The same minimum spanning tree heuristics as described in Chapter VI, Section A.3 is applied with one difference. Instead of applying it on the as-deployed SoI with full Mission Contexts and Emergent Traits, this time it is applied on the initial Capability need with partial knowledge of Intended Missions and Emergent Traits that are desired as Requirements.

Regardless of the starting state, a work plan calculated based on the Fog of Emergence is guaranteed to be shorter or equal to that of the optimal work plan. This minimized work plan results even if an Organization starts with full knowledge of the Mission Context and Emergent Traits. There are still unproductive Program Elements embedded in the workspace. These additional

Program Elements are still considered valid during planning and may give the Organization an even shorter minimum spanning tree. For Organizations starting with a smaller subset of Capability need, Intended Missions and Emergent Traits, the Organizations have a smaller set of Program Elements in their workspace and hence a shorter minimum spanning tree than the optimal.

However, starting with a shorter plan than optimal is only a transient condition because during the simulation, the transitive states would cause the Organization to pick edges that are either unproductive or not on the optimal minimum spanning tree, and incurs more and more cost in terms of its span to perform the needed set of Program Elements to accomplish the Program Activity. The only time when an Organization could complete a capability delivery at a faster rate than the optimal plan, is when it ignores Capability need inserted or failed to pick up enough negative Emergent Traits and hence deliver a sub-standard Capability.

It is inevitable that the CDSS has to prepare for a start state with a work plan shorter than the optimal, and in this implementation where we insert Program Activities into the CDS event queue based on the optimal schedules, we run into a problem where Organizations would complete their work at hand before the Program Activity is popped from the event queue for processing. As such, based on the newly calculated work plan, we insert Work Completion Events just before the scheduled Program Activities. This would allow Organizations to assess the work done and have a chance at lifting the Fog of Emergence at that point in simulation time.

Future work shall implement the other strategies, as well as varying the ratio of allocation. A one-to-one assignment of the optimal life cycle work plan could be overly generous for a Capability that starts off simple, or extremely tight for another Capability that starts off with high knowledge of Emergent Traits and Mission Contexts.

3. Run Simulation

With the CDS event queue being prepared, the simulation is ready to be run. The simulation is run simply by always popping the time sorted queue at the zeroth-element. The CDSS wraps a CdsEvent interface around the JAVA Comparable interface to allow the Java Virtual Machine to perform the time-sorting as shown in the code snippet below. This ensures that the zeroth-element of a time sorted queue is always the one with the smallest planned end in simulation time units.

```
public interface CdsEvent extends Comparable<CdsEvent> {

	/**
	 * @param pa
	 * @return -1 if this CdsEvent ends before event, 0 if ends at the same time, 1
	 *         if this CdsEvent ends after event
	 */
	public abstract int compareTo(CdsEvent event);

	/**
	 *
	 * @return the planned end date in simulation time units corresponding to
	 *         this event
	 */
	public abstract double getPlannedEnd();

}
```

In the current implementation there are three types of CDS events that implement the CdsEvent Interface. They are the Capability Insertion Events, Program Activity, and Work Completion Events. The simulation ends when the queue is empty.

It is also worthwhile to recall that the CDSS uses a cumulative schedule slippage variable as a a simplifying work around described in Chapter VI Section A.2 to avoid going through the queue to update the CDS events every time a schedule slippage occur.

a. Capability Insertion Events

(1) Update Fog of Emergence and Workspace. Recall that a Capability Insertion Event is specified by the User as described in Chapter VI Section B.2.b using the name for the Capability need to be inserted. When this event is processed, the CDSS updates the Organization's Fog of Emergence to reflect a change in the as-needed Capability.

Depending on the SEP model strategy, the Organization can choose to ignore the need, attend to it immediately, or attend to it at the next incremental iteration of the Capability. Should the Organization take up the capability need, the CDSS updates the Organization's as-planned Capability. This update triggers the unveiling of the relevant Program Elements in the various workspaces.

The current implementation of the CDSS uses the Waterfall process model which ignores changes in Capability Need. Therefore, a recommendation for future work is to incorporate the SEP models that allow for changes in Capability Need.

b. Program Activities and Work Completion Events

(1) Determine Work Done. When a Program Activity or Work Completion Event is popped from the CDS event queue, the CDSS determines how much of the current workspace has been completed.

The Work Completion Event is a special CDS event that is created to signify completion of Program Elements based on an Organization's work plan. Hence, when a Work Completion Event is processed, all the Program Elements in its work plan are considered done.

For the Program Activity, an additional check is performed to determine how much work was done. Consider an example workspace with the required Program Elements as indicated shown in Figure 38. The optimal solution would be the green minimum spanning tree with a span of 16 units.

Hence, the CDSS would have entered this Program Activity into the CDS event queue with a planned end of 16 simulation time units. However, an Organization that does not know which Program Element is unproductive, would plan to perform the blue minimum spanning tree which seems to satisfy the same requirements while costing only 11 units of simulation time. The CDSS would have scheduled that as a Work Completion Event.

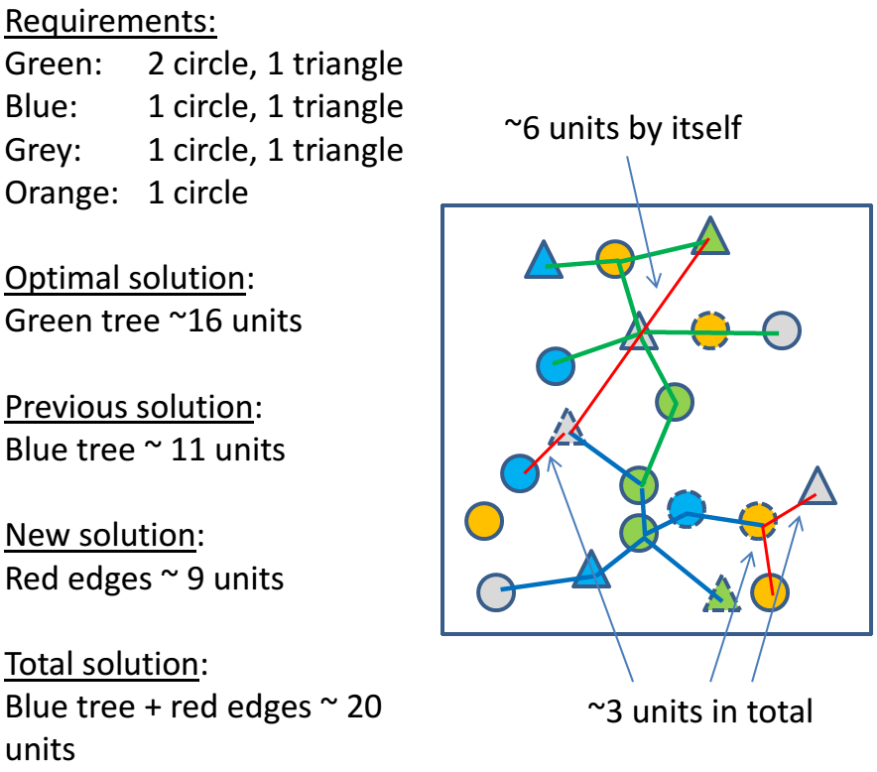


Figure 38. Work Determination Heuristics for Program Activities

However, when the Work Completion Event popped at 11 units of simulation time, the Organization would discover that it has a shortfall of one blue circle, one orange circle and one green triangle. A new work plan would be an extended minimum spanning tree that incorporates the red edges. This means that the new completion time for this work plan would be at 20 simulation time units. There is a scheduled Program Activity at 16 time units between now (11 units) and then (20 units). The implemented heuristics in the CDSS when it

encounters such a situation would be to update the work plan but not insert a new Work Completion Event. It would allow the upcoming Program Activity to pop and do a stock-take then.

When the Program Activity pops at 16 units of simulation time, the CDSS will use a greedy algorithm to determine which of the red edges have been performed. In other words, the algorithm tries to hit the most Program Elements based on the time that elapsed since the previous Work Completion Event. In this example, 5 units of simulation time would have elapsed, and using the greedy algorithm just described, the CDSS would determine that the three Program Elements that could be performed with just 3 units of simulation time are completed.

(2) Attempt to Lift Fog of Emergence. When the Program Elements that were completed were determined for Program Activities and Work Completion Events, the CDSS would check if those Program Elements were productive or not. Using the subset of productive Program Elements that were completed, the CDSS can then determine which CDS ontological entities (Operational Activities, Function, Components for example) were produced using the relationship described in Chapter VI, Section B.2.c. This successful production would be followed by a random lifting of the Fog of Emergence, as appropriate for the sub-Program Activity type using the Organization's competency and the Mission/Emergent Trait's requisite competency levels as described in Chapter VI, Section A.4.

In the current implementation of the CDSS, if the sub-Program Activity is of the type "Requirements and Analysis," this activity would need all CDS ontological entities known to the Organization to be produced before the CDSS randomizes the discovery of more unintended Mission Contexts.

If the current sub-Program Activity type is "Design and Architecture," the CDSS would allow the Organization a chance to analyze the

manifestation of Emergent Traits that are desired as Requirement for known Intended Missions for corresponding CDS ontological entity produced.

If the current sub-Program Activity type is “Development and Acquisition,” the CDSS would allow the Organization a chance to measure the manifestation of Emergent Traits beyond those expected as Requirements for known Intended Missions for corresponding CDS ontological entity produced.

If the current sub-Program Activity type is “Integration and Test,” the CDSS would allow the Organization a chance to record new Mission Contexts that were previously unintended and also for the manifestation of Emergent Traits beyond those expected as Requirement for known Intended Missions for corresponding CDS ontological entity produced.

Any newly known Mission contexts and Emergent Traits would be entered into the Organization’s subjective view through the Fog of Emergence and be utilized in their planning of future work.

(3) Update Work Space. As described before, when a new Mission or Emergent Trait is known, the corresponding Program Elements (including unproductive ones) are added to the Organization’s work space.

(4) Update Work Plan. Based on the new knowledge of Emergent Traits, the CDSS allows the Organization to determine a new work plan, similar in manner to the one described in Figure 39. That is an extended minimum spanning tree shall be calculated to cover the Program Elements needed to produce new CDS ontological entities or to suppress new Emergent traits.

It should be noted that even if an Organization did not successfully lift any of the Fog of Emergence during this event, the Organization may still need to update its work plan if it had performed unproductive Program Elements.

If the current CDS Event is a Work Completion Event, and the new work plan is scheduled to complete before the corresponding Program

Activity, a new Work Completion Event is created and inserted into the CDS event queue using that scheduled end time. If the new work plan is scheduled to complete after the corresponding Program Activity, no new CDS event would be created.

4. Record data

The current CDSS implementation uses a simple singleton Data Recorder that could be retrieved during run-time through a static accessor method. This Data Recorder allows the CDSS to record data in to separate in-memory files through a simple function call, as shown in the code snippet below.

```
DataRecorder.getInstance().record("Debug", -1.0,
        "Accessing godsViewActivityMap: "
        + godsViewActivityMap.get(key).getDescription());
```

The example above shows the CDSS writing a debug message to a Debug file, with a customized message that examines the description of a Program Activity that has an omniscient knowledge of the workspace it holds.

5. Write Output

The current implementation does not support writing of data records to XML file, but it does support the writing of data to the console as shown in a snippet of the console output below.

```
249.83805327776446    [CDS Event]: Processing Conceptual Design of type
DESIGN_AND_ARCHITECTURE
312.72378550398514    [CDS Event]: Processing Work Completion Event for MBSE
312.72378550398514    [CDS Event]: Considering productivity of packages 2 & 3
312.72378550398514    [CDS Event]: Setting workspace [3] as a known dummy
312.72378550398514    [CDS Event]: Considering productivity of packages 3 & 6
312.72378550398514    [CDS Event]: Considering productivity of packages 3 & 4
312.72378550398514    [CDS Event]: Work done up till work completion event for MBSE
       A Threshold = 10.0 Done = 6.288255432302714
       1 Threshold = 10.0 Done = 11.574258507211669
-1.0   [CDS Event]: Already known Emergent Trait exhibited by Function 1 in Mission M2:
Trait Description = 1 successfully closed gap between subjective 8.103260893602503 and
objective 8.0 by 1.8967391063974963
354.14581346649095    [CDS Event]: Inserting Cap Need 2
415.7735546678426     [CDS Event]: Processing MBSE of type DEVT_AND_ACQUISITION
490.81460380617693    [CDS Event]: Processing Work Completion Event for AoA
```

```
490.81460380617693    [CDS Event]: Considering productivity of packages 0 & 2
490.81460380617693    [CDS Event]: Considering productivity of packages 2 & 4
490.81460380617693    [CDS Event]: Considering productivity of packages 4 & 3
490.81460380617693    [CDS Event]: Work done up till work completion event for AoA
       A Threshold = 10.0 Done = 10.626537330780224
       1 Threshold = 10.0 Done = 12.162147241828698

              1_suppress_1 Threshold = 1.8967391063974972 Done = 0.0
```

VII. CONCLUSION

A. RESEARCH CONTRIBUTIONS

In summary, this thesis proposed a capability delivery ontology with the central theme of emergence and developed a CDSS prototype. Using this capability delivery ontology, the embedded fog of emergence could be used as a prism to separate the white light of capability performance into its constituent colors of “as needed,” “as-planned,” “as-known” and “as-deployed.”

While it was lamentable that the scope of research had to be reduced due increased complexity that came with the large scope captured in the literature review and time constraints encountered, the tractability of the ontology was still demonstrated through a CDSS prototype that had a partial implementation of the functionalities required of a full-fledged simulator. The CDSS prototype embodied the concepts put forward by the ontology to step through capability delivery starting with JCIDS capability need inputs and noise factors and carried it through the DAMS life cycle phases up till O&S.

This research is the first of the many steps to come, the proposed CDS ontology with Fog of Emergence provides the language construct that shows promise when used to discuss Systems Engineering issues that arise due to emergence, and also sets the stage for future designs of experiments to determine main and interaction effects between the various input and control parameters of capability delivery to determine a normative model of capability delivery with emergence. A copy of the source code for the CDSS prototype is available on request (contact Professor Gary Langford).

Future research should be mindful that the systems engineering process models are models of what their creators believe are important in the process to deliver a product, a system, or a capability. Actual capability delivery by the various organizations would inevitably pursue whatever activities needed to

deliver their system regardless of which systems engineering process they picked.

Nevertheless, experiments based on these models could uncover relative philosophical advantages or disadvantages between these models, and these insights could be used develop a normative model for capability delivery with emergence.

B. REFLECTION ON DELIVERING THE "CDSS" CAPABILITY USING THE CDS ONTOLOGY WITH FOG OF EMERGENCE

One of the research objectives of this thesis to develop a CDSS could be compared to an organization charged with delivering a capability need. The following section concludes this thesis by examining the part of the thesis journey to deliver a "capability delivery simulation" capability need, using the language constructs afforded by the CDS ontology with fog of emergence.

The capability delivery organizations involved initially assessed that the "capability delivery simulation" capability was sufficiently contained and uncomplicated and had planned to take the capability rapidly through DAMS life cycle phases of TD and E&MD to produce a working simulator capable of being used for experiments on capability delivery.

The initial analysis and requirements in Chapter III helped defined the Operational Architecture comprised of the Operational Activities, Function Flows, and Intended Missions (Use Cases) and were similar to the activities performed in preparation for the Milestone A Review of the TD phase. The quality and the comprehensiveness of the Operational Architecture were affected by the Organization's lack of requisite competency in the Discrete Event Modeling and Simulation domain. The work planned to implement the CDSS ignored a multitude of Mission Contexts (how CDS events were not just Program Activities as initially envisioned but included work completion events and capability insertion events) and unforeseen Emergent Traits (The CDS ontology with emergence has many points of articulation and each interaction between specific

instances of an entity may require some logic to be encoded) that have been lurking in the background from the very day this topic was selected.

As the Organization progressed into the Prototype Preliminary Design activities of the TD life cycle phase, the Organization completed more Program Elements and produced corresponding CDS ontological entities such as the Functional view and Component view of the Systems Architecture. With the entities produced, the Organization had the opportunity to lift the Fog of Emergence obscuring its view of the SoI. More Program Elements were revealed to the Organization with regard to more work needed to improve the preliminary design to suppress newly discovered negative Emergent Traits. An example of the negative Emergent Trait was the design choice to use a cumulative schedule slippage mechanism which effectively nixed the ability for the current CDSS prototype to allow concurrent Program Activity execution despite the fact that the Program Activity itself was fully composable as described in Chapter VI Section A.2.

In the development of the prototype, a number of Program Elements were found to be unproductive and resulted in reworked (the Component Architecture developed in Chapter V, Section A.2 was quickly found to be too rigid and hence not followed in the development). Faced with an increasing minimum spanning tree, but limited time, the original plan to take the “capability delivery simulation” capability up till the E&MD life cycle phase to produce a simulator usable for experiments, was scaled back to focus on TD activities of developing a prototype that explored the important functionalities of the CDSS to facilitate future work that involved the running of experiments. The fog of emergence had obscured the Organization’s perception of Emergent Traits in the as-deployed Mission contexts, resulting in what proved to be an overly-optimistic plan.

The saving grace was that through these ordeals, the Organization verified the tractability of the proposed CDS ontology with emergence twicefold, firstly through the CDSS, and secondly through the journey to deliver a “CDSS capability.”

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. N^2 CHARTS OF DAMS LIFE CYCLE PHASES

Figure 39 below shows the high level overview of the N^2 chart of the five DAMS life cycle phases: (1) Materiel Solutions Analysis (MSA); (2) Technology Development (TD); (3) Engineering & Manufacturing Development; (4) Production & Deployment (P&D); and (5) Operations & Support (O&S).

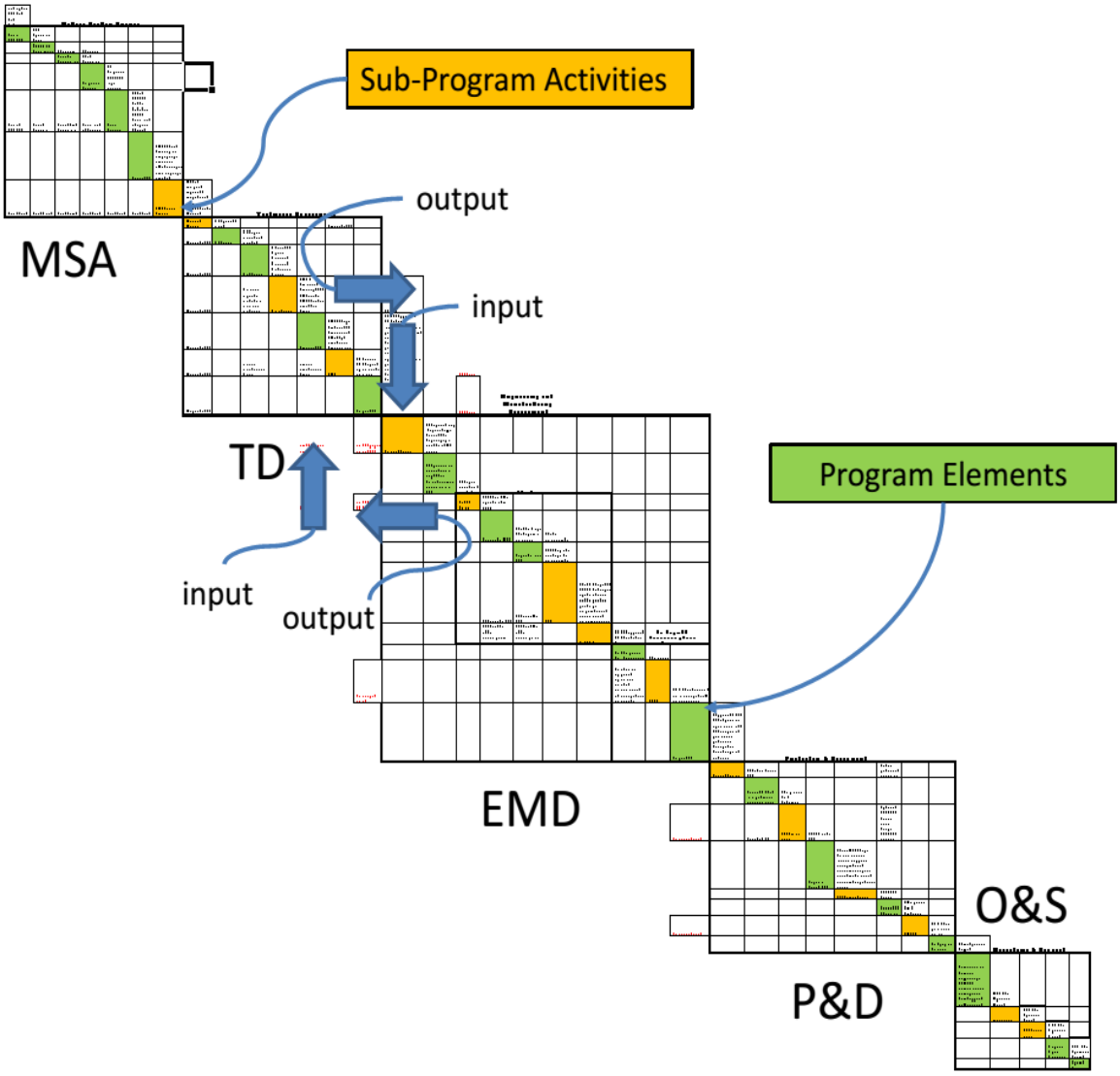


Figure 39. Overview of N^2 Charts of DAMS Life cycle Phases.

The diagonal spine of the chart is formed by the generic sequence of program elements (in green cells) required to produce products, systems architecture and SoI components (as input and output arrows) needed to accomplish program activities (in yellow cells). The following five tables follow the same convention and provide a zoomed in view of each DAMS life cycle phase in more detail.

Table 14. N^2 Chart Materiel Solution Analysis.

Initial Capability Doc, MDD, AoA Study Guidance	**Materiel Solution Analysis**					
Define CONOPS	[O&M] Operational Activities					
	Functional Decomposition	[F]Functions	[F] Functions			
		Value System Development	[R] MoEs, Requirements			
			Component Allocation	[SA] Component allocation for multiples alternatives		
Refined CONOPS	Refined Functions	Refined MoEs, Requirements	Recommended Alternative	Analysis of Alternatives	[SA] AoA: CONOPS, MoEs, Cost, Schedule, CTEs, Risk, Recommended options to ICD needs	

Initial Capability Doc, MDD, AoA Study Guidance		**Materiel Solution Analysis**					
					Prepare TDS	[SA] TDS: Tech development strategy, single-step or evolutionary, schedules, cost, goals, increments, prototypes needed	
Feedback	Feedback	Feedback	Feedback	Feedback	Feedback	MSA Review Milestone	[SA] AoA completed, options for ICD capability needs recommended, and TDS ready for Milestone A

Table 15. N^2 Chart Technology Development.

[SA] AoA completed, options for ICD capability needs recommended, and TDS ready for Milestone A		**Technology Development**				
Milestone A Review	[SA] Options for ICD needs				Planned in TDS	
Cost growth > 25%	RFP process	[SA] Proposal selected and awarded				

[SA] AoA completed, options for ICD capability needs recommended, and TDS ready for Milestone A		**Technology Development**					
Cost growth > 25%		Build Prototypes	[C] Refined SA (Especially Function & Process) and Production of Prototypes				
Cost growth > 25%		Demonstrate again: Not affordable, militarily useful, mature technology	Demo Prototypes	[O&M + C] Demonstrated: Develop LCSP, SEP (includes RAMS) based on candidate designs			[SA] SEP

Cost growth > 25% Cost growth > 25%		Insufficient confidence for design		Prepare for PDRs Insufficient confidence for design	[SA] PDR Report: Hw, Sw and HIS baseline, refined SA with high confidence design at system-level PDR	[SA] Successful PDR Report with requirement trades, cost estimation	[SA] CDD Approved by JROC = {Tech & manufacturing processes for program/increment identified and SoI can be developed for production within 5 years (usually), contains key operational performance parameters}, Refined Integrated Architecture, Clarification plan to become a warfighting capability, LRIP quantities (one unit to 10% of total)
Cost growth > 25%						Prepare CDD	

Table 16. N^2 Chart Engineering & Manufacturing Development (From Integrated Systems Design).

[SA] CDD Approved by JROC = {Tech & manufacturing processes for program/increment identified and SoI can be developed for production within 5 years (usually), contains key operational performance parameters}, Refined Integrated Architecture, Clarification plan to become a war fighting capability, LRIP quantities (one unit to 10% of total)		PDR Report PDR Report	**Engineering and Manufacturing Development**				
Milestone B Review	[SA] Acquisition Strategy + Acquisition Program Baseline (APB) = Acquisition program initiated, Minimal LRIP quantities						

	RFP process (ensure no award to offerors using CTEs not demoed in relevant environment + no TRL)	[SA] Proposal selected and awarded	**Integrated System Design**				
		Post-PDR Assessment	[SA] MDA inform PM of required remedial actions				
			Prepare sub-system CDR	[C] Sub-System designs & building of some components	[C] Sub-system components built		
				Prepare System-level CDR	[C} CDR Report: System-level design & list of components built		
			PM direct to redo CDR	PM direct to redo CDR	CDR	[C] Post CDR Report: SME & CDR Chair, Description of product baseline and %age of built-to packages completed, issues and actions for closure, risk assessment against exit criteria	
			MDA direct PM to address resolution / mitigation plans	MDA direct PM to address resolution/mitigation plans		Post CDR Assessment	[SA] MDA approved CDR & Initial Product Baseline

Table 17. N^2 Chart Engineering & Manufacturing Development (From System Capability & Manufacturing Process Demonstration).

	[SA] MDA approved CDR & Initial Product Baseline	**System Capability & Manufacturing Process Demonstration**		
	Build Components (System + Manufacturing)	[C] Components		
New capability needs	Unable to meet approved requirements in intended environment and industrial capabilities are not available	DT&E	[O&M+C] Satisfied user needs in terms of mission capability and operational support	
			Prepare CPD	[C] Approved by JROC: CPD = {Operational requirements informed by EMD results, expected performance of production system}, Acceptable -ilities, Refined integrated architecture

Table 18. N^2 Chart Production & Deployment

[C] Approved by JROC: CPD = {Operational requirements informed by EMD results, expected performance of production system}, Acceptable -ilities, Refined integrated architecture	**Production & Deployment**						
Milestone C Review	[SA] Authorized entry into LRIP				Authorize production & procurement		
	Execute LRIP, build minimum production-representative articles	[C] Components (System & Production)					
	Not ready for FRP	IOT&E in mission context	[C&OA] Ready for FRP		If (skip beyond LRIP & FRP decision review) Congress, USD(AT&L) approve		
			Prepare for beyond LRIP	[C] Beyond LRIP Report: Demonstrate control of manufacturing process, acceptable reliability, collection of statistical process control data, demonstrated control and capability of critical processes			

				FRP Decision Review	[C] MDA's FRP Decision		
					Execute FRP & Deployment	[C] Components (System & Production)	
						FOT&E	[O&M+C] Mission performance assessment
							Military Equipment Valuation

Table 19. N^2 Chart Operations & Support

[SA] Initial Operational Capability	**Operations & Support**			
Life cycle sustainment: Continual engineering for RAMS, HSI, environmental safety, occupational health, supportability, and interoperability	[O&M+C] Full Operational Capability			
	Iterative reviews	[O&M+C] Full Operational Capability		
		PEO Annual reviews	[O&M+C] Full Operational Capability	
			Prepare for Disposal / Repurpose	[O&M+C] Full Operational Capability
				Disposed / Repurposed

APPENDIX B: CAPABILITY DELIVERY SYSTEM SIMULATOR USE CASES

A. USE CASE TEMPLATES

1. "Start Simulator"

Use Case: Start simulator

Primary actor: User

Goal in context: To start the simulator and load the input file.

Preconditions:

- CDSS must be compiled and the executable set on the computer's PATH variable
- Input file specifying input and control parameters must be well-formed
- Destination folder must exist and with "write" permission enabled
- User has brought up the command line interface

Trigger: Intention to use the CDSS.

Scenario:

- The user enters command to run program with two parameters specifying the location of input file and the destination folder/name of output file:
- Java CDSS [input file location] [output file destination]
- The software loads the input file and displays a summary on the input and control parameters read in.

Exceptions:

- Incorrect number of command parameters. The software displays an example of an expected command and exits.
- Input file is not well formed.

Priority: Moderate priority, to be implemented as second increment.

When Available: Prototype 2

Frequency of use: High frequency

Channel to actor: Via PC-based command line interface to link the software with input files.

Open Issues:

- What input format would facilitate less error-prone specification of input and control parameters?
- What output file format would facilitate ease of analysis through a separate spreadsheet program such as Excel?

2. "Enter Simulator Mode"

Use Case: Enter simulator mode

Primary actor: User

Goal in context: To set simulator mode to either print event trail on the console or to just run with no event trail.

Preconditions:

- User has already performed Use case "Start simulator."

Trigger: The CDSS presents the user with three options as listed below:

"1 – Exit simulator; 2- Display event trail on console; [Anything else] - No event trail on console."

Scenario:

- If user enters "1," the CDSS shall skip simulation and exits.
- Else if user enters "2," the Console displays "Event trail mode set" and begins to run the simulator.
- Else if user enters anything else (including empty return), the Console displays "Mode: No Event Trail" and begins to run the simulator.

Exceptions: NA

Priority: Moderate priority, to be implemented as second increment

When Available: Prototype 2

Frequency of use: High frequency

Channel to actor: Via PC-based command line interface to link the Software with input file

Open Issues: NA

3. "Run Simulator"

Use Case: Run simulator

Primary actor: User

Goal in context: To run simulator and to generate output file containing simulation results based on provided input and control parameters.

Preconditions:

- User must has already performed use case "Set simulator mode"

Trigger: The CDSS received a simulation mode.

Scenario:

- The CDSS starts the discrete event simulation.
- If "Event Trail," the CDSS display status update on the Console for every discrete event until simulation is over.
- Else if "No Event Trail," the CDSS insert a period ."“on the Console display every 3s to indicate that it is running until simulation is over.
- The software writes the full event trail and results into the Output File.
- The software displays a summary of the output parameters via the Console and terminates.

Exceptions:

Output destination does not exist. Software shall inform the user via the console and attempt to write output file to local folder as "Output.csv."

Priority: Top priority, to be implemented immediately.

When Available: Prototype 1

Frequency of use: High frequency

Channel to actor: Via PC-based command line interface to link the software with input file.

Open Issues:

What output file format would facilitate ease of analysis through a separate spreadsheet program such as Excel?

LIST OF REFERENCES

Bent, K., Grenning, J., Martin, R. C., Beedle, M., Highsmith, J., Mellor, S., & Marick, B. (2001). *Manifesto for Agile software development*. Retrieved Jul 2013, from http://agilemanifesto.org/.

Boehm, B., & Lane, J. A. (2007). Using the incremental commitment model to integrate system acquisition, systems engineering, and software engineering. *CrossTalk*, *20*(10), 4–9. Retrieved on date, from http://www.crosstalkonline.org/storage/issue-archives/2007/200710/200710–0-Issue.pdf.

Boehm, B. (2000). *Spiral development: Experience, principles, and refinements*. W.J. Hansen (Ed.). Retrieved from Carnegie Mellon University, Software Engineering Institute website: Retrieved Jul 2013, from http://www.sei.cmu.edu/library/abstracts/reports/00sr008.cfm.

Center for Technology in Government (CTG) (1998). *A survey of system development process models* (CTG. MFA - 003). Retrieved from State University of New York at Albany website: Retrieved May 13, from http://www.ctg.albany.edu/publications/reports/survey_of_sysdev/survey_of_sysdev.pdf.

Dahmann, J., Lane, J. A., Rebovich, G., & Baldwin, K. J. (2008). A model of systems engineering in a system of systems context. *Paper presented at the Sixth Conference on Systems Engineering Research (CSER)*. Retrieved May 2013, from http://www.acq.osd.mil/se/docs/2008–04–04_CSER-Paper_Dahmann-etal-SoS.pdf.

Dahmann, J. S., Rebovich Jr., G. R., & Lane, J. A. (2008). *Systems engineering for capabilities*. Retrieved Jun 2013 from Crosstalk: The Journal of Defense Software Engineering website: http://www.crosstalkonline.org/storage/issue-archives/2008/200811/200811-Dahmann.pdf.

Dahmann, J., Rebovich, G., Lowry, R., & Baldwin, K. (2011). *An implementer's view of systems engineering for systems of systems*. doi:10.1109/SYSCON.2011.5929039.

Defense Acquisition University (DAU) (2008). *Materiel development decision*. Retrieved Jul 2013, from http://acc.dau.mil/ils_mdd.

Deputy CIO (2010). *DoDAF glossary*. Retrieved Jun 2013 from U.S. Department of Defense website: http://dodcio.defense.gov/dodaf20/dodaf20_glossary.aspx.

DoD CIO (2012). *DoDAF version 2.0 plenary*. Retrieved Jul 2013 from U.S. Department of Defense website: http://dodcio.defense.gov/Portals/0/Documents/DODAF/DoDAF_v2–02_web.pdf

Fact sheet: The ballistic missile defense system. (2013). Fort Belvoir, Virginia: Missile Defense Agency.

Forsberg, K., & Mooz, H. (1995). *The relationship of systems engineering to the project cycle*. Cupertino, California: Center for Systems Management.

Fruhling, A. L., & Tarrel, A. E. (2008). *Best practices for implementing Agile methods: A guide for Department of Defense software developers*. Retrieved from IBM Center for the Business of Government website: http://www.businessofgovernment.org/report/best-practices-implementing-agile-methods-guide-department-defense-software-developers

Hitchins, D. K. (2000). *World class systems engineering–the 5-layer model*. Retrieved Jul 2013, from http://www.hitchins.net/systems/world-class-systems-enginee.html

Holland, J. H. (1998). *Emergence: From chaos to order*. Reading, Mass: Addison-Wesley.

International Council on Systems Engineering (INCOSE) (2010). *Systems engineering handbook v3.2*. Hampton, VA: INCOSE.

Jamshidi, M. (2009). *System of systems engineering: Innovations for the 21st century*. Hoboken, NJ: Wiley.

JCIDS Manual. (2012). Retrieved Jun 2013 from http://jitc.fhu.disa.mil/jitc_dri/pdfs/jcids_manual_19jan12.pdf

Kasser, J. E. (2012). *Complex solutions for complex problems*. Retrieved May 2013 from Third International Engineering Systems Symposium, CESUN 2012, Delft University of Technology, June 18–20, website: http://cesun2012.tudelft.nl/images/3/33/Kasser.pdf

Kasser, J. E. (2010). *Seven systems engineering myths and the corresponding realities*. Retrieved Apr 2013 from Proceedings of the Systems Engineering Test and Evaluation Conference (SETE 2010), Adelaide, Australia website: http://www.therightrequirement.com/pubs/2010/myths%20of%20systems%20engineering-5.pdf

Keating, C. B. (2005). Research foundations for system of systems engineering. In *Proceedings of IEEE International Conference Systems on Man and Cybernetics*, *3* (pp. 2720–2725). doi:10.1109/ICSMC.2005.1571561

Keet, M. K. (2008). *A formal theory of granularity.* (Doctoral dissertation). Retrieved Aug 2013 from http://www.meteck.org/PhDthesis.html

Koolmanojwong, S. (2010). *The incremental commitment spiral model process patterns for rapid-fielding projects*. (Doctoral dissertation). Retrieved Aug 2013 from http://csse.usc.edu/csse/TECHRPTS/PhD_Dissertations/files/Koolmanojwong_Dissertation.pdf.

Langford, G. O. (2012). *Engineering systems integration: Theory, metrics, and methods*. Boca Raton: CRC Press.

Langford, G. O. (2013a). *SE4151 Systems Engineering Integration* [PowerPoint slides].

Langford, G. O. (2013b). *SE3100 Fundamentals of Systems Engineering* [PowerPoint slides].

Maier, M. W., & Rechtin, E. (2009). *The art of systems architecting* (3rd ed.). Boca Raton: CRC Press.

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, *1*(4), 267–84.

McBreen, P. (2002, February 8). *Software Development: Dismantling the Waterfall | Some Flaws | InformIT*. Retrieved from http://www.informit.com/articles/article.aspx?p=25272

Nogueira, J. C., Jones, C., & Luqi (2000). *Surfing the Edge of Chaos: Applications to Software Engineering*. Retrieved Aug 2013 from Command and Control Research and Technology Symposium, Naval Postgraduate School, Monterey, CA website: http://www.dodccrp.org/events/2000_CCRTS/html/pdf_papers/Track_4/075.pdf

Office of Deputy Under Secretary of Defense for Acquisition & Technology, Systems and Software Engineering (2006). *Guide for integrating systems engineering into DoD acquisition contracts version 1.0*. Washington, DC: ODUSD (A&T) SSE.

Office of Deputy Under Secretary of Defense for Acquisition & Technology, Systems and Software Engineering (2008). *Systems engineering guide for systems of systems version 1.0*. Washington, DC: ODUSD (A&T) SSE.

Office of Under Secretary of Defense for Acquisition, Technology, and Logistics (OUSD AT&L) (2008). *Department of Defense Instruction Number 5000.02*. Washington, DC.

Osmundson, J. A., Huynh, T. V., & Langford, G. O. (2007). System of systems management issues.

Pressman, R. S. (2010). *Software engineering: A practitioner's approach*. New York: McGraw-Hill Higher Education.

Pyster, A., & Olwell, D. H. (2013). The guide to the systems engineering body of knowledge (SEBoK), v.1.1.2. *The Trustees of the Stevens Institute of Technology* [Hoboken, NJ]. Retrieved Jun 2013 from www.sebokwiki.org

Quadrennial defense review report. (2001). Washington, D.C.: U.S. DoD.

Rechtin, E. (1991). *Systems architecting: Creating and building complex systems*. Englewood Cliffs: Prentice Hall.

Royce, W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*, *26*, 1–9. Retrieved Aug 2013 from http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

Schrader, J. Y., Lewis, L., & Brown, R. A. (2003). *Quadrennial defense review 2001: Lessons on managing change in the Department of Defense*. Santa Monica, CA: RAND.

United States Department of Defense (USDoD) (1988). *DoD-STD-2167A, Military standard: Defense system software development*. Retrieved Jul 2013 from http://www.everyspec.com/DoD/DoD-STD/DoD-STD-2167A_8470/

Vitech (2011). *CORE 8 architecture definition guide*. Retrieved Jul 2013 from Vitech Corporation website: http://www.vitechcorp.com/support/documentation/core/800/ArchitectureDefinitionGuide.pdf

Wolpert, D. H. (2008). Physical limits of inference. *Physica D-nonlinear Phenomena*, *237*(9), p. 1257–1281. Retrieved Aug 2013 from dx.doi.org/10.1016/j.physd.2008.03.040

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California